

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

До захисту допущено:

В.о. завідувача кафедри

(підпис) Олександр ПАВЛОВ
(вл.ім'я, прізвище)

“ ____ ” _____ 2020 р.

Дипломний проєкт
на здобуття ступеня бакалавра

**за освітньо-професійною програмою «Інформаційні управляючі
системи та технології»
спеціальності 122 «Комп'ютерні науки та інформаційні технології»**

на тему: «Інформаційна система складання планів
обслуговування терміналів з прийому платежів у населення»

Виконала: студентка IV курсу, групи ІС-61

Лукова Оксана Юріївна
(прізвище, ім'я, по батькові) (підпис)

Керівник

доц., к.т.н., доц. Жданова Олена Григорівна
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

**Консультант з
графічної
документації**

доц., к.т.н., доц. Телишева Тамара Олексіївна
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент

доц., к.т.н., доц. Репнікова Наталія Борисівна
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цьому дипломному проєкті
немає запозичень з праць інших авторів без
відповідних посилань.

Студентка _____
(підпис)

Київ – 2020 року

**Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет (інститут) інформатики та обчислювальної техніки
(повна назва)

Кафедра автоматизованих систем обробки інформації і управління
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки та інформаційні технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

Олександр ПАВЛОВ
(підпис) (вл.ім'я, прізвище)

“ ” 2020 р.

**ЗАВДАННЯ
на дипломний проєкт студенту**

Луковій Оксані Юріївні
(прізвище, ім'я, по батькові)

1. Тема проєкту «*Інформаційна система складання планів
обслуговування терміналів з прийому платежів у населення*»

керівник проєкту Жданова Олена Григорівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “7”травня 2020 р. №1081-с

2. Термін подання студентом проєкту “01”червня 2020 року

3. Вихідні дані до проєкту

Технічне завдання

4. Зміст пояснювальної записки

1. *Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі*

2. *Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних*

3. *Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв'язання*

4. *Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів*

5. *Технологічний розділ: керівництво користувача, методика випробувань програмного продукту*

5. Перелік графічного матеріалу

1. Схема структурна діяльності
2. Схема бази даних
3. Схема структурна класів програмного забезпечення
4. Схема структурна послідовності
5. Схема структурна компонентів програмного забезпечення
6. Схема модифікованого алгоритму метода найближчого сусіда
7. Схема алгоритму метода бджолиного рою

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Загальні положення	доц. Жданова О.Г.		
Інформаційне забезпечення	доц. Жданова О.Г.		
Математичне забезпечення	доц. Жданова О.Г.		
Програмне та технічне забезпечення	доц. Жданова О.Г.		
Технологічний розділ	доц. Жданова О.Г.		

7. Дата видачі завдання «13» квітня 2020 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення рекомендованої літератури	16.04.2020	
2.	Аналіз існуючих методів розв'язання задачі	23.04.2020	
3.	Постановка та формалізація задачі	23.04.2020	
4.	Розробка інформаційного забезпечення	30.04.2020	
5.	Алгоритмізація задачі	05.05.2020	
6.	Обґрунтування використовуваних технічних засобів	10.05.2020	
7.	Розробка програмного забезпечення	15.05.2020	
8.	Налагодження програми	01.06.2020	
9.	Виконання графічних документів	01.06.2020	
10.	Оформлення пояснювальної записки	01.06.2020	
11.	Подання ДП на попередній захист	15.05.2020	
12.	Подання ДП на основний захист	01.06.2020	
13.	Подання ДП рецензенту	02.06.2020	

Студент

Оксана ЛУКОВА

Керівник

Олена

ЖДАНОВА

[illegible]

Пояснювальна записка до дипломного проєкту

на тему: Інформаційна система складання планів обслуговування
терміналів з прийому платежів у населення

Київ – 2020 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту складається з п'яти розділів, містить 29 рисунка, 26 таблиць, 1 додаток, 3 джерел.

Дипломний проєкт присвячений розробці комплексу задач складання планів обслуговування терміналів з прийому платежів у населення. Метою створення системи є зменшення витрат на інкасацію терміналів з прийому платежів за рахунок складання оптимальних або близьких до них планів інкасації терміналів та прогнозування наповненості терміналів. Для досягнення поставленої мети потрібно виконати наступні задачі:

- прогнозування наповненості терміналів готівкою, яка надходить від клієнтів – користувачів терміналів;
- складання оптимальних або близьких до них планів інкасації терміналів, яке враховує результати прогнозування наповненості терміналів.

У розділі інформаційного забезпечення були визначенні вхідні та вихідні дані та документи системи, визначено таблиці бази даних та побудована схема бази даних.

У розділі математичного забезпечення було представлено постановку задачі маршрутизації транспортних засобів VRP та методи її розв'язання для двох випадків. Для методів було створено узагальнені схеми алгоритмів, застосовано модифікації класичних алгоритмів для можливості їх паралельної реалізації, що дозволить швидше отримувати результати, зберігаючи точність розрахунків, або за той же час отримувати кращі результати.

					ДП 6117.00.000 ПЗ			
<i>Зм.</i>	<i>Арк.</i>	<i>Прізвище</i>	<i>Підпис</i>	<i>Дата</i>	Інформаційна система складання планів обслуговування терміналів з прийому платежів у населення	<i>Лім.</i>	<i>Лист</i>	<i>Листів</i>
<i>Розроб.</i>		Лукова О.Ю					2	
<i>Перевірив.</i>		Жданова О.Г						
<i>Н. кон.</i>		Телишева Т.О.				КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-61		
<i>Затв.</i>								

У розділі програмного забезпечення обґрунтовано обрані засоби розробки. ^{Навчов 94} Python, ReactJS, Postgres. Для кожної технології розглянуто переваги та недоліки у розрізі конкретного проекту. У розділі також наведено діаграму класів та опис кожного класу, а також детальний опис усіх функцій програми. Наведено діаграми послідовності та діаграми компонентів даного проекту.

У технологічному розділі наведено інструкцію користувача з описаними діями, які можна виконувати, описана послідовність роботи в залежності від дій. Також наведено випробування системи, набори тестів, що дозволяють перевірити правильність роботи системи.

ЗАДАЧА МАРШРУТИЗАЦІЇ ТРАНСПОРТНИХ ЗАСОБІВ, ЗАДАЧА КОМІВОЯЖЕРА, МЕТОД НАЙБЛИЖЧОГО СУСІДА, МЕТОД БДЖОЛИНИХ КОЛОНІЙ

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of five sections, contains 29 figures, 26 tables, 1 appendices, 3 sources.

The project is devoted to the development of a set of tasks for planned maintenance of cash payment terminals for payment acceptance. The purpose of developing the system is to reduce the cost of planned maintenance of cash payment terminals for payment acceptance optimal or close to them plans for collection of terminals and forecasting the fullness of terminals. To achieve this goal you need to perform the following tasks:

- to analyze the existing methods of solving the problem of making plans;
- develop a solution algorithm;
- to develop a software implementation of the algorithm;
- to test the effectiveness of the developed algorithms.

In the section of information support the input and output data and documents of the system were determined, the database tables were determined and the database scheme was constructed.

In the section of mathematical support, the problem of routing VRP vehicles and methods of its solution for two cases were presented. Generalized schemes of algorithms were created for methods. Modifications of classical algorithms were applied for possibility of their parallel realization that will allow to receive results faster, keeping accuracy of calculations, or at the same time to receive the best results.

The software section substantiates the selected development tools: Python, ReactJS, Postgres. For each technology, the advantages and disadvantages of a specific project are considered. The section also provides a class diagram and description of each class, as well as a detailed description of all program functions. Sequence diagrams and component diagrams of this project are given.

The technological section provides user instructions with the described actions that can be performed, describes the sequence of work depending on the

					ДП 6117.00.000 ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

actions. There are also tests of the system, sets of tests to verify the correctness of the system.

THE VEHICLE ROUTING PROBLEM, TRAVELING SALESMAN PROBLEM, THE NEAREST NEIGHBOR METHOD, THE METHOD OF BEE COLONIES

					ДП 6117.00.000 ПЗ	Арк.
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

<u>ВСТУП</u>	15
<u>1 ЗАГАЛЬНІ ПОЛОЖЕННЯ</u>	17
<u>1.1 ОПИС ПРЕДМЕТНОГО СЕРЕДОВИЩА</u>	17
<u>1.1.1 Опис процесу діяльності</u>	17
<u>1.1.2 Опис функціональної моделі</u>	18
<u>1.2 ОГЛЯД НАЯВНИХ АНАЛОГІВ</u>	20
<u>1.3 ПОСТАНОВКА ЗАДАЧІ</u>	21
<u>1.3.1 Призначення розробки</u>	21
<u>1.3.2 Цілі та задачі розробки</u>	21
<u>Висновок до розділу</u>	22
<u>2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ</u>	23
<u>2.1 ВХІДНІ ДАНІ</u>	23
<u>2.2 ВИХІДНІ ДАНІ</u>	24
<u>2.3 ОПИС СТРУКТУРИ БАЗИ ДАНИХ</u>	27
<u>Висновок до розділу</u>	28
<u>3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ</u>	29
<u>3.1 ЗМІСТОВНА ПОСТАНОВКА ЗАДАЧІ</u>	29
<u>3.2 МАТЕМАТИЧНА ПОСТАНОВКА ЗАДАЧІ</u>	30
<u>3.3 ОБГРУНТУВАННЯ МЕТОДУ РОЗВ'ЯЗАННЯ</u>	31
<u>3.4 ОПИС МЕТОДІВ РОЗВ'ЯЗАННЯ</u>	32
<u>3.4.1 Опис методу найближчого сусіда</u>	32
<u>3.4.2 Опис алгоритму бджолиних колоній</u>	32
<u>3.4.3 Результати роботи методу найближчого сусіда</u>	34
<u>3.4.4 Результати роботи методу бджолиних колоній</u>	35
<u>Висновок до розділу</u>	39
<u>4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ</u>	40
<u>4.1 ЗАСОБИ РОЗРОБКИ</u>	40
<u>4.2 ВИМОГИ ДО ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ</u>	41
<u>4.2.1 Загальні вимоги</u>	41
<u>4.3 АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ</u>	41

4.3.1	<u>Діаграма класів</u>	41
4.3.2	<u>Діаграма послідовності</u>	42
4.3.3	<u>Діаграма компонентів</u>	43
4.3.4	<u>Специфікація функцій</u>	44
4.4	<u>ОПИС ЗВІТІВ</u>	52
	<u>Висновок до розділу</u>	55
5	<u>ТЕХНОЛОГІЧНИЙ РОЗДІЛ</u>	56
5.1	<u>КЕРІВНИЦТВО КОРИСТУВАЧА</u>	56
5.2	<u>ВИПРОБУВАННЯ ПРОГРАМНОГО ПРОДУКТУ</u>	62
5.2.1	<u>Мета випробувань</u>	62
5.2.2	<u>Загальні положення</u>	62
5.2.3	<u>Результати випробувань</u>	62
	<u>Висновок до розділу</u>	64
	<u>ЗАГАЛЬНІ ВИСНОВКИ</u>	65
	<u>ПЕРЕЛІК ПОСИЛАНЬ</u>	66
	<u>ДОДАТОК А</u>	67

ВСТУП

Кожен з нас користується кожного дня готівкою та постійно використовує термінали та банкомати самообслуговування для отримання готівки або поповнення карти для безготівкового розрахунку. Найчастіше, коли ми потребуємо послуг терміналів або банкоматів, вони знаходяться у неробочому стані через надлишок готівки (для терміналів) або недостатчу готівки (для банкоматів).

Коренем цієї проблеми є неправильна організація інкасування як зі сторони банків, так і зі сторони інкасаторських служб. Банки не розраховують необхідну кількість готівки, що їм потрібна, і на який час буде її достатньо для коректної роботи терміналів та банкоматів для задоволення попиту користувачів. Інкасаційні служби їдуть на інкасацію за деяким графіком з фіксованою періодичністю. Через це виникають декілька проблем: одні термінали наповнюються швидше та перестають працювати, інші не наповнюються навіть до половини, але на них також витрачаються гроші для інкасації, хоча вони могли б працювати ще до наступної інкасації.

Вирішення цієї проблеми повинно проходити у декілька етапів. Перший - це вирішення питання «На який період вистачить місця у терміналі?» і друге - «За яким маршрутом потрібно інкасувати термінали?».

Перше питання можна вирішити завдяки статистичному моделюванню, а друге розв'язанням задачі маршрутизації транспортних засобів [1].

Головною частиною роботи - є задача маршрутизації транспортних засобів (Vehicle Routing Problem – VRP), тобто складання близьких до оптимальних планів обслуговування клієнтів. Задача маршрутизації є NP-складною, а отже потребує значних витрат на розрахунки, задля забезпечення більш точних результатів. Отже, є потреба у побудові алгоритмів, що здатні отримувати наближені до точних результатів за мінімально можливий час [2].

					ДП 6117.00.000 ПЗ	Арк.
						15
Змн.	Арк.	№ докум.	Підпис	Дата		

Дипломний проєкт присвячений розробці комплексу задач складання планів інкасації терміналів з прийому платежів, який допоможе з вирішенням прогнозування періодичності інкасацій та складання планів інкасацій для терміналів, що потребують інкасації.

Практичне значення одержаних результатів. Розроблено алгоритми розв'язання задачі маршрутизації для одного транспортного засобу методів найближчого сусіда та методом бджолиних колоній. Зроблене їх порівняння за часом та відхилення від оптимального результату.

Публікації. Результати роботи були опубліковані у 1 тезах доповідей на науково-технічних конференціях [2].

					ДП 6117.00.000 ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Є банк, що складається з:

- головного відділення;
- відділень, що підпорядковуються головному;
- терміналів для прийому платежів, що підпорядковуються одному з відділень.

Кожне відділення повинно інкасувати свої термінали вчасно аби термінали не зупиняли свою роботу, коли в них закінчується місце для готівки. Для цього банки користуються послугами інкасаторських служб та проводять планові інкасації за попередньо складеним розкладом.

Ця система призводить до простоїв деяких терміналів та неповного заповнення інших терміналів. Інkasатори проводять інкасації одразу всіх терміналів, на що витрачається більша кількість грошей, аніж можна було витратити. Тому подібна система потребує автоматизації та оптимізації процесів.

Оптимізацію можна провести шляхом прогнозування наповненості терміналів та складання більш економних маршрутів інкасаторської машини по заздалегідь визначеним терміналам.

Необхідно провести прогнозування наповненості терміналів та визначення у який день потрібно проводити інкасацію кожного, визначити найбільш вигідний день інкасації та скласти маршрут інкасації на кожен день.

1.1.1 Опис процесу діяльності

За допомогою UML- діаграми діяльності опишемо дії, які повинні виконуватись у процесі роботи з системою.

					ДП 6117.00.000 ПЗ	Арк.
						17
Змн.	Арк.	№ докум.	Підпис	Дата		

На початку роботи з системою банківський аналітик завантажує файли до системи, інформація з яких додається до бази даних: список усіх відділень та терміналів банку, історичні дані щодо наповненості терміналів, інформація про банк.

Кожного наступного дня аналітик повинен давати новий файл про наповненість терміналів за попередній день. Система щоранку завантажує інформацію з нового файлу у базу даних та відображає її на інтерфейсі.

Після оновлення всіх даних, на інтерфейсі аналітик отримує повідомлення щодо успішності оновлення даних на поточний день. Коли дані вже оновлені, то аналітик може продивити поточні та прогнозовані наповненості терміналів, обираючи потрібний день на календарі.

Також аналітик може запустити побудову маршруту. Система, використовуючи дані з бази даних почне розрахунки: виконить прогнозування наповненості та побудову маршрутів на наступні два дні, якщо такі маршрути будуть потрібні, та отримає повідомлення про успішне завершення побудови маршрутів. Після побудови маршрутів можна буде побачити список усіх маршрутів, що виконувались до цього, та прогнозованих маршрутів, які можна відмінити, якщо такий не потрібен. На мапі буде відображено маршрут на поточний день.

Дані, отриманні при роботі системи, використовуються для формування звітності та покращення аналізу у системі та відображаються у формі графіків.

Діаграма діяльності наведена у графічному матеріалі.

1.1.2 Опис функціональної моделі

Актором у діаграмі варіантів використання є банківський аналітик.

Банківський аналітик відповідає за завантаження нових даних, аналіз вихідних даних системи, які може доповнювати або видаляти за

необхідністю, ведення обліку наявних відділень та терміналів банку та ведення інкасаторських машин, що працюють.

Банківський аналітик може запускати розрахунки на побудову нових маршрутів, обираючи частість проведення перевозок, та переглядати отримані системою результати.

Аналітик може передивитись можливі перевезення на найближчі 2 дні та вартість маршрутів, а також історичну завантаженість терміналів та прогнозовану на наступні два дні.

Також аналітик може проглянути звіти з результатами роботи системи та графіки порівняння результатів періодичних перевезень та перевезень виконаних з використанням системи.

Система повинна виконувати наступні функції:

- ведення наявних терміналів;
- ведення обліку інкасаторських машин;
- прогнозування наповненості терміналів готівкою;
- складання планів інкасації терміналів;
- формування звітності.

Діаграма варіантів використання наведена на рисунку 1.1.

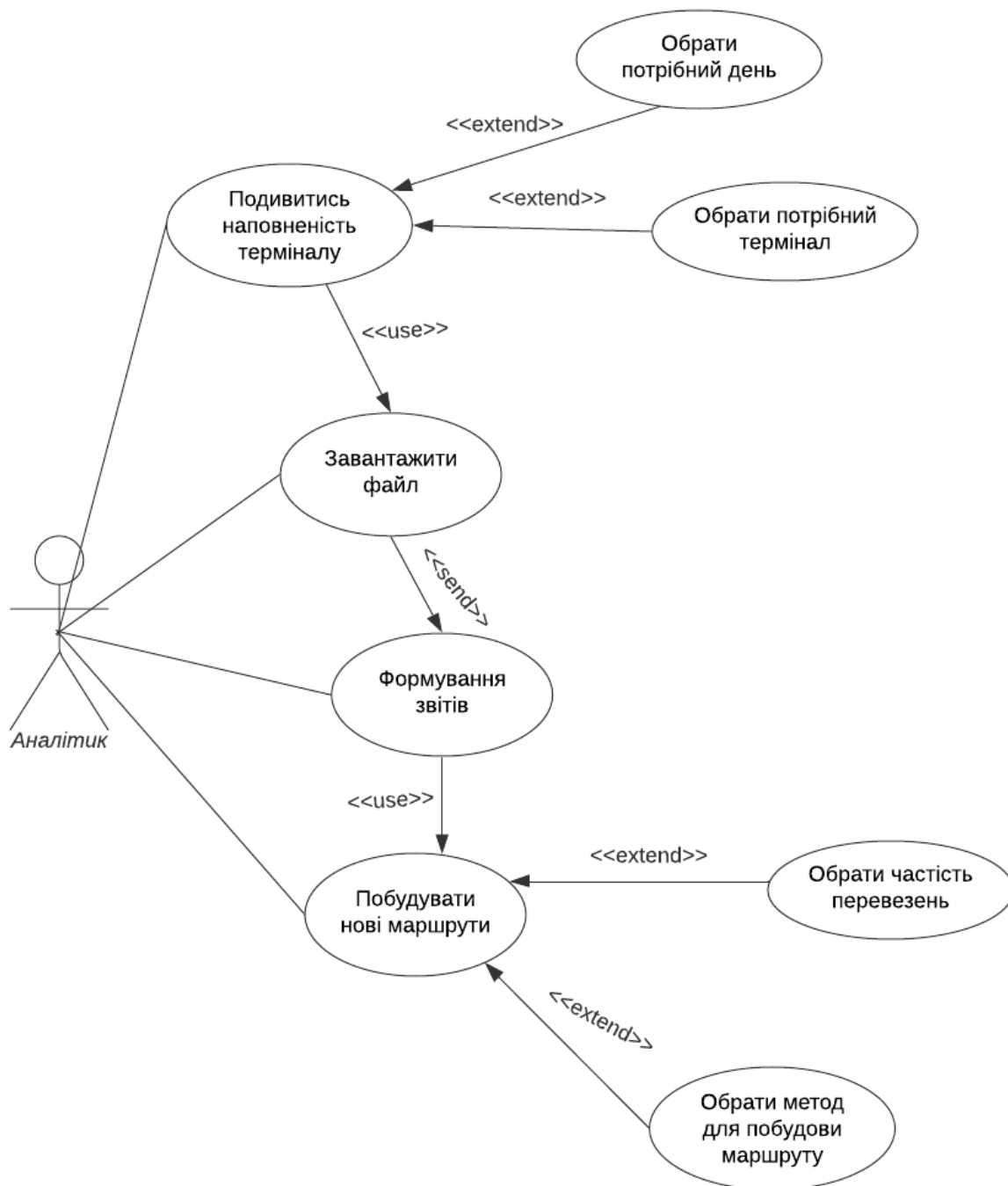


Рисунок 1.1 – Схема структурна варіантів використання

1.2 Огляд наявних аналогів

На момент написання роботи даних про реалізовані програмні продукти з автоматизацією цих процесів в Україні немає. Проте була знайдена презентація компанії SAS, яка презентувала свої інноваційні кейси у сфері банківської аналітики. Один з кейсів – «Оптимізація інкасації

банкоматів та терміналів» є близьким за ідеєю та вирішує ряд питань таких як:

- скільки грошей потрібно покласти у кожний термінал;
- які та скільки банкнот повинно бути;
- як часто можливе поповнення;
- яка послідовність інкасації.

Вирішення цих питань складається з двох етапів:

- прогнозування зняття та поповнення;
- оптимізація інкасацій.

За попередніми розрахунками проект може отримати скорочення витрат на інкасації у 2-3 рази.

1.3 Постановка задачі

1.3.1 Призначення розробки

Призначенням розробки є складання планів інкасації терміналів з прийому платежів.

1.3.2 Цілі та задачі розробки

Метою створення системи є зменшення витрат на інкасацію терміналів з прийому платежів за рахунок складання оптимальних або близьких до них планів інкасації терміналів та прогнозування наповненості терміналів.

Для досягнення поставленої мети потрібно виконати наступні задачі:

- прогнозування наповненості терміналів готівкою, яка надходить від клієнтів – користувачів терміналів;
- складання планів інкасації терміналів з мінімальною вартістю реалізації, яке враховує результати прогнозування наповненості терміналів.

Висновок до розділу

У розділі було проаналізовано предметне середовище, розроблено процес діяльності, за яким повинна працювати система, визначені функції користувачів та розроблена діаграма діяльності, визначенні призначення та цілі розробки системи.

					ДП 6117.00.000 ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Вхідні дані – це дані які система отримує для своєї роботи.

Вхідні дані вносяться в систему перед початком її використання. Дані про наповненість терміналів за попередній день додаються до системи кожного дня вранці.

Вхідними даними системи є:

- дані про відділення та термінали;
- дані про вартості перевезень;
- дані про наповненість терміналів;
- дані про банк.

У таблиці 2.1 наведені реквізити відповідних документів.

Таблиця 2.1 – Реквізити документів вхідних даних

№	Найменування	Реквізити
1	Дані про відділення та термінали	назва; тип; назва батьківського відділення (для терміналів); адреса (координати).
	Дані про вартості перевезень	вартість виїзду одного транспортного засобу на маршрут; вартість проїзду між терміналами та відділеннями.
3	Дані про наповненість терміналів	дата; назва; процент наповненості на відповідний момент часу.
4	Дані про банк	назва банку; адреса розміщення головного відділення; валюта, що використовується.

2.2 Вихідні дані

Вихідними даними системи є:

- план перевезень на вказаний горизонт часу;
- вартість та довжина визначених планів;
- прогнозована наповненість терміналу на вказаний горизонт часу;
- звіт про виконані перевезення.

Макети екранних форм наведені на рисунках 2.1 – 2.4.

На рисунку 2.1 наведено макет головної сторінки, яка відображає основні результати: список планів перевезень на вказаний горизонт часу, вартість та довжина визначених планів та прогнозована наповненість терміналів.

На рисунку 2.2 наведено макет побудованого маршруту на мапі, який можна продивитись обираючи відповідний день на календарі.

На рисунку 2.3 наведено макет інформаційного вікна з прогнозованою наповненість терміналу, який можна подивитись обираючи відповідний день на календарі та термінал на мапі.


На рисунку 2.4 наведено макет сторінки зі звітами, на якій буде відображено звіти про виконані перевезення та витрати у вигляді графіків.

Page 1

https://localhost

Bank Analytics

Upload File



October 2014

Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4
5	6	7	8	9	10	11

☐ Every day

Run Analytics Report

Date	Lenght	Cost	Delete
2020-05-05	1000	5000	
2020-05-06	1500	6000	

Рисунок 2.1 – Макет головної сторінки



Рисунок 2.2 – Приклад макету побудови маршруту



Рисунок 2.3 – Приклад макету інформаційного вікна з прогнозованою наповненість термінала

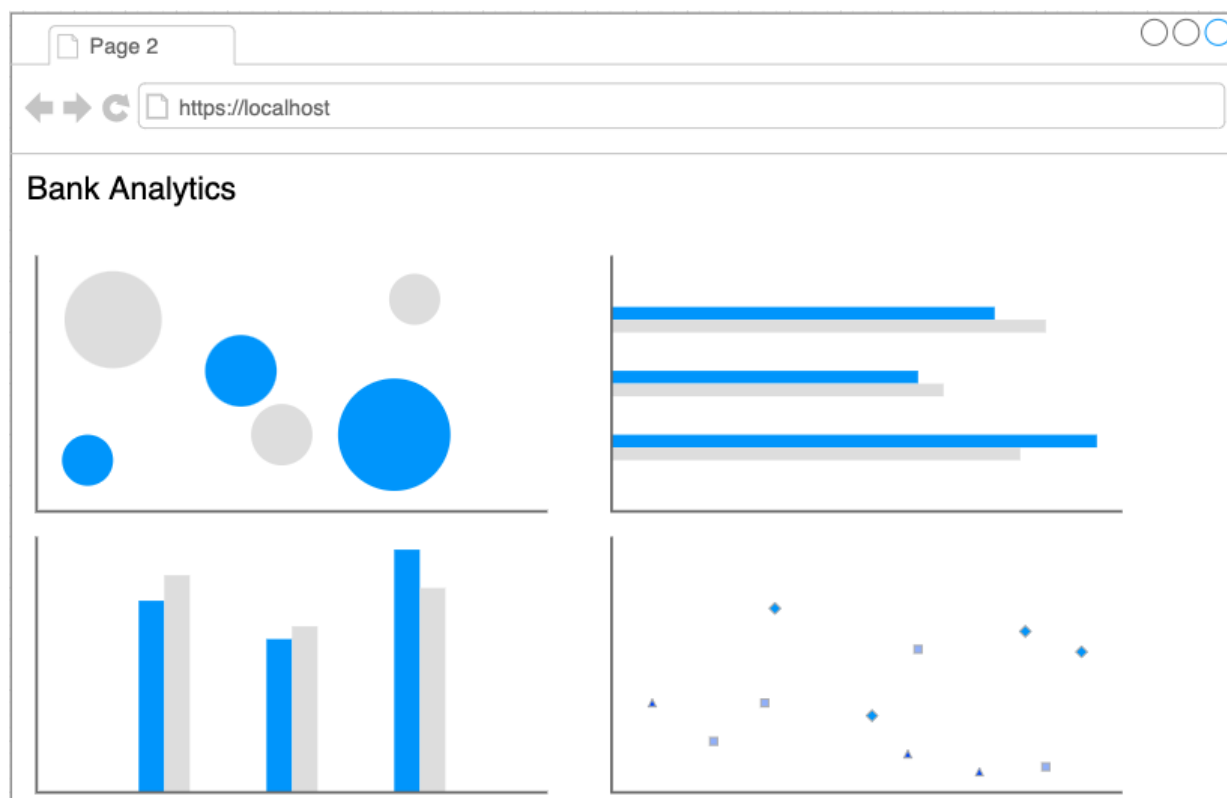


Рисунок 2.4 – Макет сторінки зі звітами

Змн.	Арк.	№ докум.	Підпис	Дата

2.3 Опис структури бази даних

У таблицях 2.2 – 2.7 наведена структура таблиць бази даних.

Таблиця 2.2 – Структура таблиці cash_entities (таблиця відділень та терміналів)

Назва поля	Тип даних	Опис поля
ce_code	INTEGER	Ключовий ідентифікатор відділення/терміналу
ce_name	TEXT	Назва відділення/терміналу
ce_type	TEXT	Тип (відділення/термінал)
parent_code	INTEGER	Ключовий ідентифікатор батьківського відділення
x_coord	TEXT	Координата по осі абсцис
y_coord	TEXT	Координата по осі ординат

Таблиця 2.3 – Структура таблиці balances (таблиця балансів терміналів)

Назва поля	Тип даних	Опис поля
date	DATE	Дата балансів
code	INTEGER	Ключовий ідентифікатор терміналу
percent	DECIMAL	Процент наповненості на поточну дату

Таблиця 2.4 – Структура таблиці shipments (таблиця перевезень)

Назва поля	Тип даних	Опис поля
date	DATE	Дата маршруту
point	INTEGER []	План маршруту
length	DECIMAL	Довжина маршруту
cost	DECIMAL	Вартість маршруту

Таблиця 2.5 – Структура таблиці shipment_durations (таблиця вартостей)

Назва поля	Тип даних	Опис поля
from_code	INTEGER	Ідентифікатор відділення/терміналу звідки їде транспортний засіб
to_code	INTEGER	Ідентифікатор відділення/терміналу куди їде транспортний засіб
distance	INTEGER	Довжина переїзду
duration	INTEGER	Тривалість переїзду

Таблиця 2.6 – Структура таблиці forecast_balances (таблиця прогнозованих балансів терміналів)

Назва поля	Тип даних	Опис поля
date	DATE	Дата прогнозованого балансів
code	INTEGER	Ключовий ідентифікатор терміналу
percent	DECIMAL	Прогнозований процент наповненості на поточну дату

Таблиця 2.7 – Структура таблиці forecast_entity_shipment (таблиця прогнозованих інкасації терміналів)

Назва поля	Тип даних	Опис поля
ce_code	INTEGER	Ключовий ідентифікатор терміналу
next_date	DATE	Прогнозована дата інкасації

Схема бази даних наведена у графічному матеріалі.

Висновок до розділу

У розділі були визначенні вхідні та вихідні дані та документи системи, визначено таблиці бази даних та побудована схема бази даних.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Змістовна постановка задачі

У місті є n відділень банку та m терміналів. Кожен з терміналів належить тільки одному з відділень, тобто задано вектор p належності терміналу:

Відома транспортна мережа , яка з'єднує термінали, де - не порожня скінчена множина вершин (терміналів); - множина ребер: , кожному ребру мережі поставлена у відповідність невід'ємна вага (вартість) та час проходження ребра [2].

Приклад розташування терміналів та відділень наведено на рисунку 3.1.

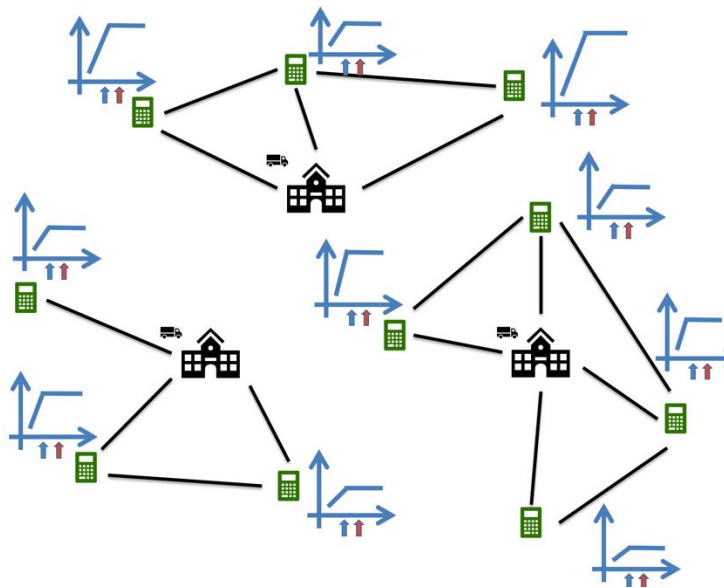


Рисунок 3.1 – Приклад розташування терміналів та відділень

Термінали приймають готівку від користувачів і заповнюються при накопиченні готівки в кількості S (в загальному випадку поріг накопичення терміналу становить) . Після заповнення термінал припиняє роботу і відключається до моменту вивезення з нього готівки. Такі вимушені

простої призводять до зниження рентабельності мережі терміналів через зменшення грошових надходжень в банк через термінали. Банківські відділення мають у своєму розпорядженні парк інкасаторських машин, які періодично направляються для вивезення готівки зі своїх терміналів. Кожен виїзд інкасаторської машини має певну вартість, яка залежить від типу машини, часу знаходження на маршруті та пройденої відстані [2].

Відділення збирають статистичні дані щодо інтенсивності заповнення кожного із своїх терміналів, щоденно зберігаючи інформацію про поточний стан (заповнений/незаповнений) та про кількість грошей, що надійшли до терміналу на кінець робочого дня. З урахуванням цієї інформації розрахована середня інтенсивність надходження грошей до кожного терміналу

(грн/день). Відповідно цього розраховані величин

і, де J - збитки терміналу за одну годину вимушеного простою (грн/год) [2].

Інкасаторські машини можуть працювати тільки у банківські дні. Якщо термінал буде заповнений у банківські вихідні, то він буде відключений («залишиться без роботи та заробітку») до наступної інкасації. А якщо ж вивозити готівку занадто рано, то це збільшує витрати на обслуговування терміналів, призводить до зниження прибутковості мережі терміналів [2].

Скласти розклад виїзду інкасаторських машини, таким чином щоб мінімізувати сумарні витрати на інкасацію грошей з терміналів та збитки від можливих вимушених простоїв [2].

3.2 Математична постановка задачі

Виходячи з позначень постановки задачі, маємо класичну задачу маршрутизації, де граф $G = (V, E)$, V – множина вершин, E – множина ребер. Кожному ребру $e \in E$ у відповідність

					ДП 6117.00.000 ПЗ	Арк.
						30
Змн.	Арк.	№ докум.	Підпис	Дата		

поставлена вартість . Індексом позначатимемо відповідний транспортний засіб (де – множина транспортних засобів) [2].

(3.1)

(3.2)

(3.3)

(3.4)

(3.5)

(3.6)

Цільова функція (3.1) визначає вартість маршрутів для всіх транспортних засобів. Обмеження (3.2) гарантує, що всі клієнти будуть відвідані тільки одним транспортним засобом та тільки один раз. Обмеження (3.3) та (3.4) гарантують, що транспортний засіб покидає та повертається в депо тільки один раз. Обмеження (3.5) гарантує, що коли транспортний засіб прибуває до вершини h , то він повинен з неї виїхати [2].

3.3 Обґрунтування методу розв'язання

В якості головних алгоритмів системи були обрані метод найближчого сусіда та метод бджолиного рою.

Класичний метод найближчого сусіда є жадібним алгоритмом. Ідея алгоритму найближчого сусіда заснована на евристичному правилі: якщо на кожному кроці відвідувати найближчий пункт, то в цілому маршрут також вийде досить непоганий. Алгоритм забезпечує недопущення повторного заїзду до пункту та передчасного повернення в початковий пункт [2].

Метод бджолиного рою є евристичним методом випадкового пошуку. Метод є одним з новіших та перші ідеї його застосування були опубліковані у 2005р [3]. Ідея методу у моделюванні поведінки бджіл під час пошуку нектару.

Процес пошуку нектару починається з пошуку більш перспективних ділянок для збору нектару. Рій бджіл поділяється на декілька типів бджіл один з яких бджоли-розвідники. Бджоли-розвідники випадковим чином шукають перспективні ділянки, повертаються до вулика та інформують своїх родичів щодо якості кожної ділянки. Бджоли-розвідники відкладають нектар та виходять на танцювальний майданчик перед вуликом щоб проінформувати інших бджіл виконуючи “waggle dance”.

Два методи будуть порівнюватись для отримання кращого результату.

3.4 Опис методів розв’язання

3.4.1 Опис методу найближчого сусіда

Для удосконалення результатів алгоритму було введено модифікацію, яка дозволяє покращити отримувані розв’язки без додаткових витрат часу. Це досягається за рахунок паралельної реалізації алгоритму [2].

Якщо першою точкою маршруту обирати кожен з наданих, або за деяким правилом, то алгоритм буде покривати більшу кількість варіантів маршрутів та зможе знайти кращий варіант маршруту, ніж при випадковому виборі однієї стартової точки [2].

Перший пункт обирається зі списку терміналів, що потребуються інкасації [2].

Схема алгоритму наведена у графічному матеріалі.

3.4.2 Опис алгоритму бджолиних колоній

Модель поведінки бджіл полягає у наступному. Спочатку бджоли-розвідники вилітають випадковим чином на пошук нектару. Після

					ДП 6117.00.000 ПЗ	Арк.
						32
Змн.	Арк.	№ докум.	Підпис	Дата		

знаходження перспективних ділянок бджоли-розвідник повертаються до вулика та повідомлять про знаходження активних бджіл. Активні бджоли вилітають на перспективні ділянки та локально збирають нектар, знаходячи або ще більш кращі ділянки або гірші.

Параметри алгоритму:

- кількість бджіл-розвідників;
- кількість бджіл, що полетять на кращі ділянки;
- кількість бджіл, що полетять на обрані ділянки;
- кількість кращих ділянок;
- кількість обраних ділянок (наступні декілька ділянок після кращих);
- кількість повторів при незмінній поточно рекордній фітнес-функції;
- максимальна кількість циклів роботи алгоритму.

Схема алгоритму наведена у графічному матеріалі.

3.4.2.1 Створити бджіл

На початку алгоритму кожна бджола отримує випадкову ділянку, на якій буде проводити локальний пошук. На цьому етапі також обирається поточна найкраща ділянка, з якою буде проводитись подальше порівняння.

3.4.2.2 Знайти нові ділянки

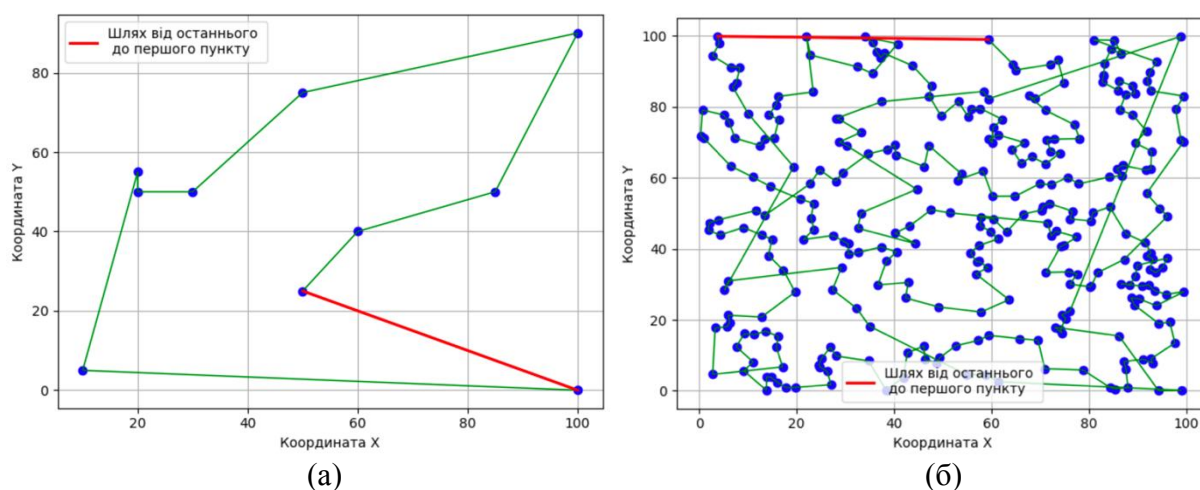
На цьому етапі бджоли локально шукають нові ділянки, проводячи локальні модифікації своєї поточної ділянки та порівнюють цей результат з глобальним найкращим.

3.4.2.3 Танець бджіл

Цей етап проходять бджоли, коли знаходять результат кращий за поточно рекордний для того, щоб наступні бджоли почали працювати на кращих та обраних ділянках.

3.4.3 Результати роботи методу найближчого сусіда

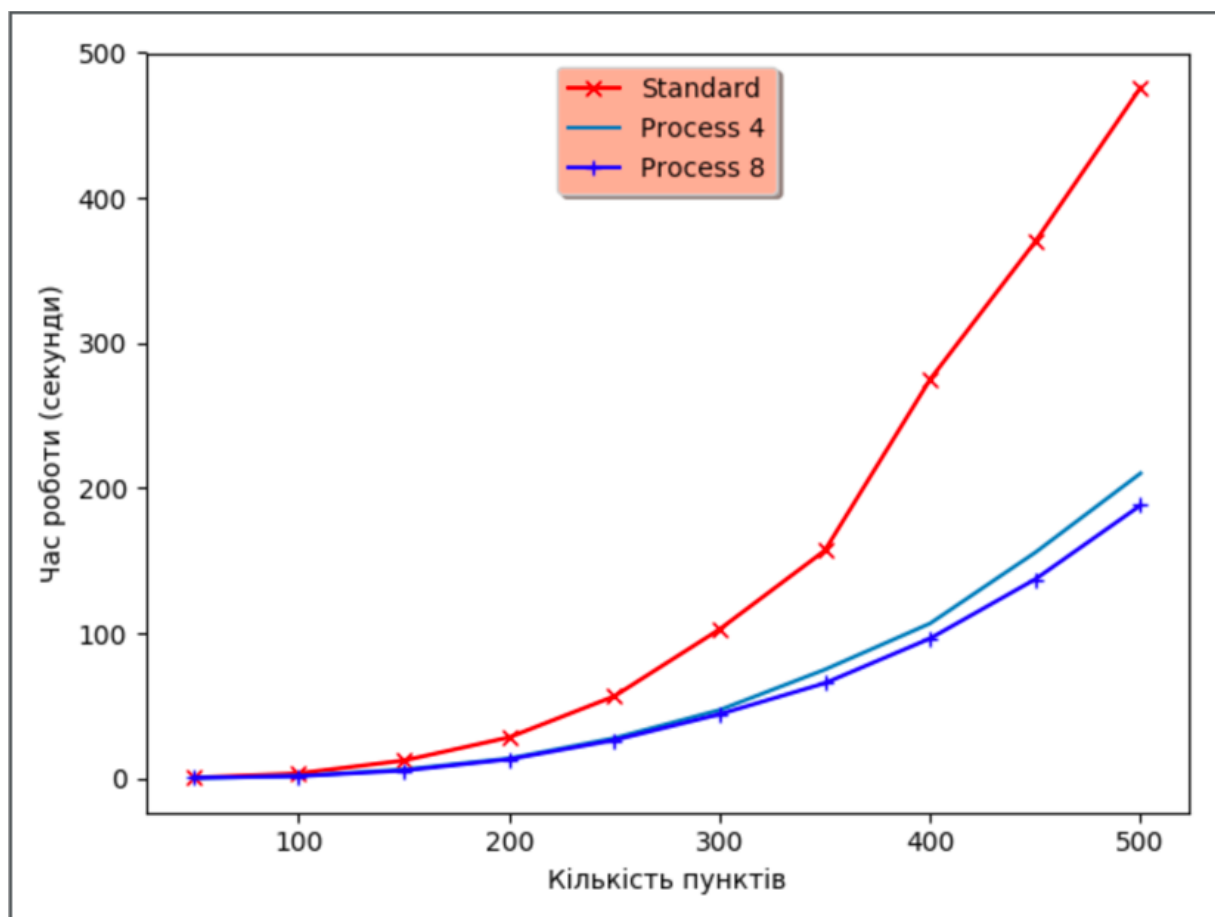
В результаті роботи алгоритму були отримані доволі точні результати. Були проведені експерименти на 10 та 300 пунктах обслуговування. На графіках зображені пункти обслуговування з координатами, що відмічені по осі абсцис та осі ординат та отриманні шляхи між ними (рисуюнок 3.2 а, б) [2].



Рисуюнок 3.2 – Результат роботи методу найближчого сусіда (а, б)

Наведені результати порівняння часу роботи послідовної та паралельної реалізації алгоритму. Кількість пунктів обслуговування по осі абсцис та час виконання у секундах по осі ординат (рисуюнок 3.3), де Standart – час роботи послідовної реалізації алгоритмі, Process4 – час роботи паралельної реалізації на 4 процессах, Process8 – час роботи паралельної реалізації на 8 процессах. Для вирішення проблеми великих витрат по часу було використано паралельна реалізація модифікованого алгоритму, що дозволило отримувати кращі результати за більш короткий проміжок часу. Тому для подальших досліджень будуть розроблятись саме такі алгоритми які дозволяють робити їх паралельні реалізації [2].

Змн.	Арк.	№ докум.	Підпис	Дата



Standart – час роботи послідовної реалізації алгоритмі, Process4 – час роботи паралельної реалізації на 4 процеслах, Process8 – час роботи паралельної реалізації на 8 процеслах [2].

Рисунок 3.3 – Графік порівняння часу роботи послідовної та паралельної реалізації методу найближчого сусіда

3.4.4 Результати роботи методу бджолиних колоній

По результатам роботи алгоритму була отримана хороша збіжність за досить малу кількість ітерацій, що дає можливість для подальшого покращення алгоритму для вирішення задачі. Приклад збіжності алгоритму наведено на рисунку 3.4.

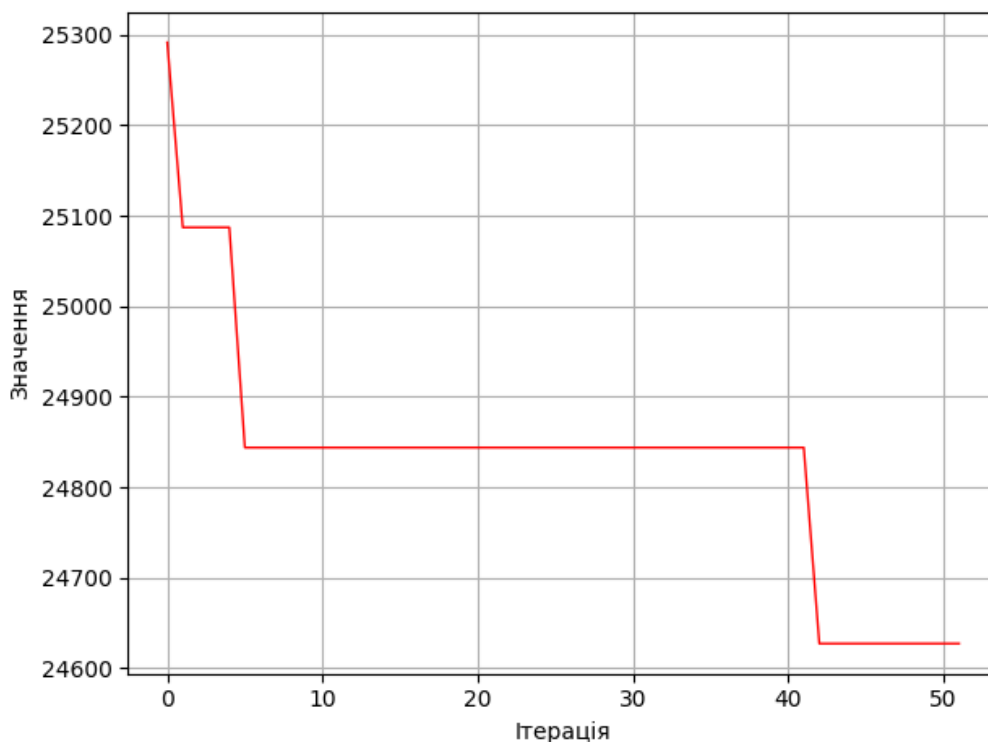


Рисунок 3.4 – Приклад збіжності алгоритму для 100 пунктів обслуговування за 50 ітерацій

Для оцінки роботи алгоритму були порівняні три методи: метод повного перебору, метод найближчого сусіда та метод бджолиних колоній. Метод найближчого сусіда та метод бджолиних колоній порівнювались за критерієм часу та відхиленням від оптимального результату, отриманого з методу повного перебору.

Порівняння часу виконання та отриманого результату наведено у таблиці 3.1.

Відхилення довжин маршрутів наведені у таблиці 3.2.

Таблиця 3.1 – Таблиця порівняння часу виконання для трьох методів

Кількість пунктів	Метод повного перебору		Метод найближчого сусіда		Метод бджолиних колоній	
	Довжина маршруту	Час роботи	Довжина маршруту	Час роботи	Довжина маршруту	Час роботи
8	295.9354	0.0355	326.9829	0.0267	295.9354	0.139
9	311.055	0.261	334.160	0.0261	318.686	0.140
10	298.808	2.489	326.057	0.0327	301.599	0.1631

Продовження таблиці 3.1

Кількість пунктів	Метод повного перебору		Метод найближчого сусіда		Метод бджолиних колоній	
	Довжина маршруту	Час роботи	Довжина маршруту	Час роботи	Довжина маршруту	Час роботи
11	257.188	26.7674	289.153	0.0271	283.662	0.166
12	300.875	335.048	360.337	0.0281	334.880	0.1735

Таблиця 3.2 – Таблиця порівняння відхилення результату від оптимального

Кількість пунктів	Метод повного перебору	Метод найближчого сусіда		Метод бджолиних колоній	
	Довжина маршруту	Довжина маршруту	Відхилення	Довжина маршруту	Відхилення
8	295.9354	326.9829	10.49%	295.9354	0%
9	311.055	334.160	7.42%	318.686	2.4%
10	298.808	326.057	9.11%	301.599	0.9%
11	257.188	289.153	12.42%	283.662	10.29%
12	300.875	360.337	19.76%	334.880	11.3%

На рисунку 3.5 наведено порівняння результатів отриманих кожним алгоритмом.

На рисунку 3.6 наведено приклад побудови маршруту для кожного з алгоритмів та збіжність алгоритму для метода бджолиних колоній на конкретному прикладі.

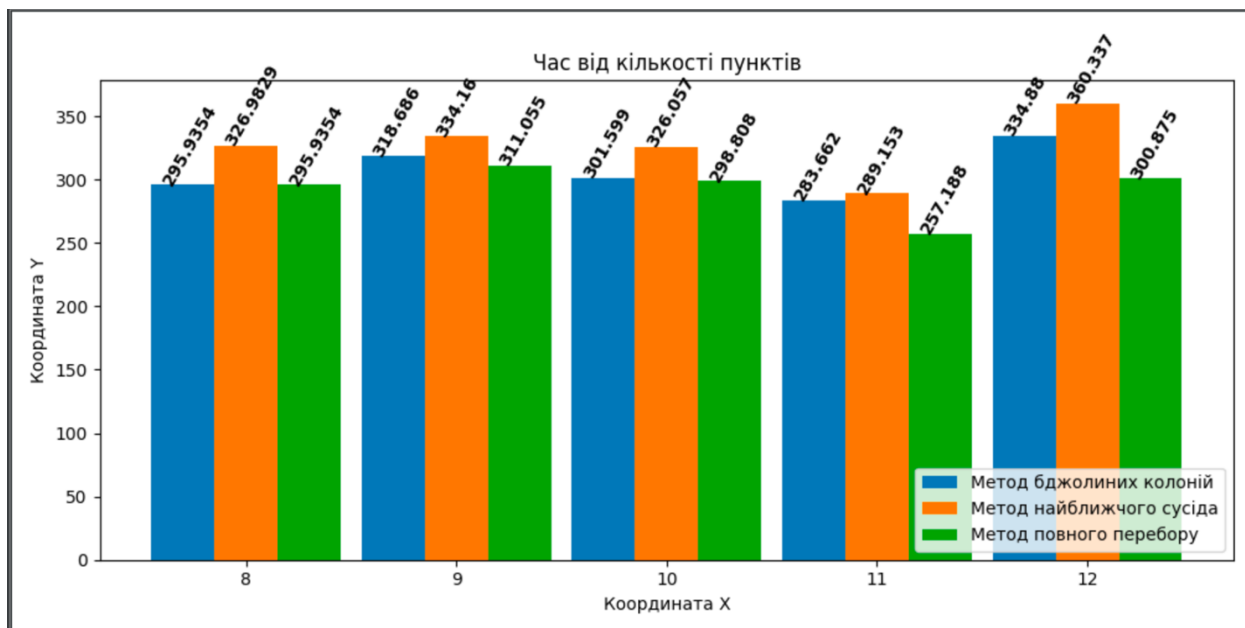


Рисунок 3.5 – Порівняння часу роботи алгоритмів

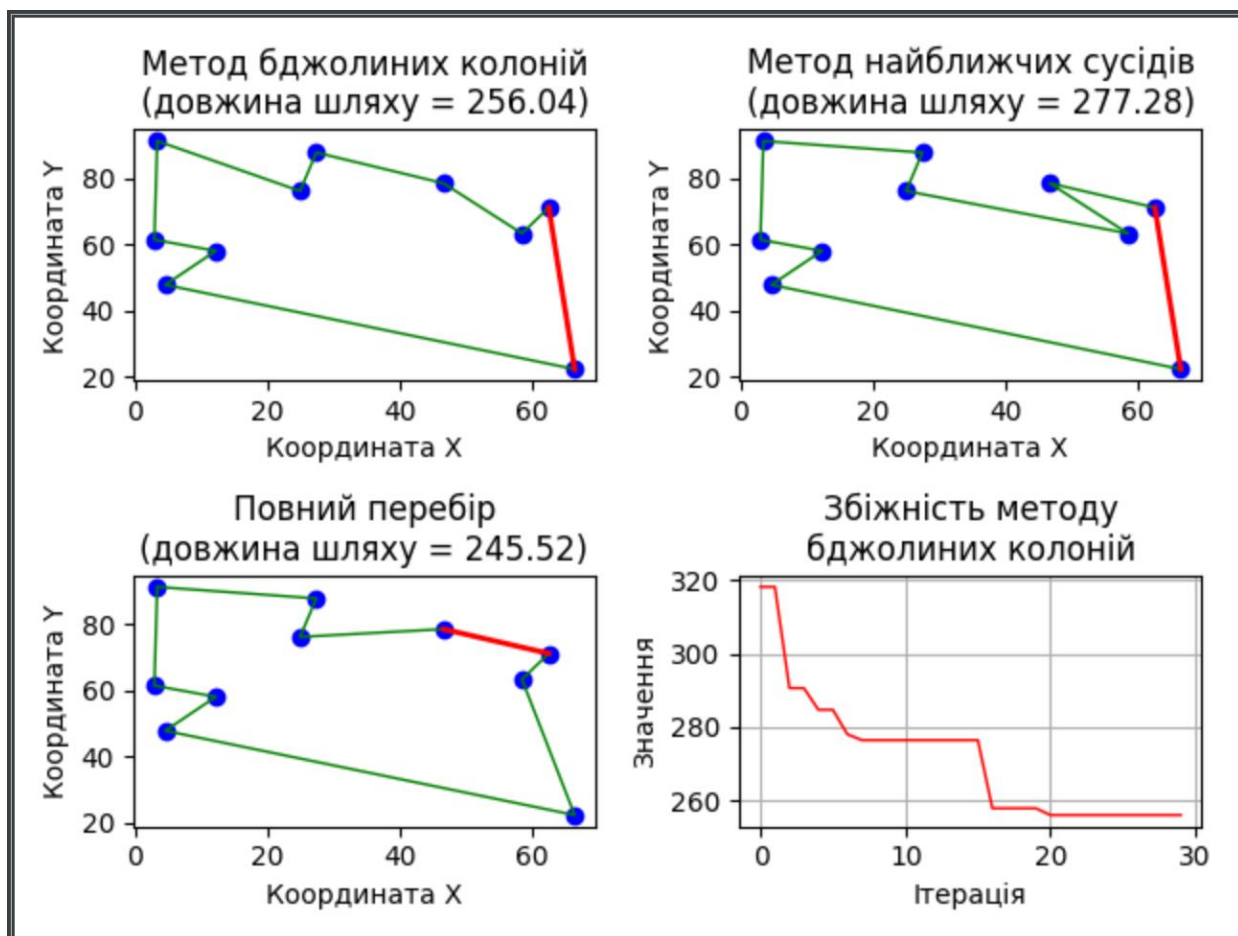


Рисунок 3.6 – Приклад результуючих маршрутів

Змн.	Арк.	№ докум.	Підпис	Дата

Порівняння проводилось на досить малих кількостях пунктів для того, щоб отримати точні результати за допомогою методу повного перебору.

Як можна побачити з таблиці 3.1, час роботи алгоритму для методу повного перебору зростає дуже швидко, тому не має сенсу запускати його для більшої кількості пунктів.

Метод бджолиних колоній показує себе досить добре. Він отримує кращі результати за метод найближчого сусіда, проте програє йому у часі, але сильно виграє у часі в порівнянні з методом повного перебору.

Метод бджолиних колоній має доволі багато параметрів, які впливають як на час роботи, так і на результат. І чим більше часу займає його робота, тим краще стає результат, тому має сенс збільшувати час роботи алгоритму, покращуючи результати.

Так як метод бджолиних колоній працює довше, аніж метод найближчого сусіда, то має сенс також покращувати час роботи методу, що можна досягти завдяки паралельній реалізації алгоритму.

В результаті можна отримати менший час роботи та кращі результати на більш великих кількостях пунктів.

Висновок до розділу

У розділі було представлено постановку задачі маршрутизації транспортних засобів VRP та методи її розв'язання для двох випадків. Для методів було створено узагальнені схеми алгоритмів. Було застосовано модифікації класичних алгоритмів для можливості їх паралельної реалізації, що дозволить швидше отримувати результати, зберігаючи точність розрахунків, або за той же час отримувати кращі результати.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

Для створення програмного продукту були застосовані мови програмування Python 3 та фреймворк мови JavaScript – ReactJS.

У даному продукту Python використовується для розробки бекенд частини та аналітичних розрахунків. В першу чергу вибір мови обґрунтовується швидкістю розробки на ньому. Python містить велику кількість стандартних бібліотек для роботи зі структурами даних, має бібліотеки NumPy та Pandas для наукових обчислень та швидкого маніпулювання даними.

Найважливішими перевагами Python є:

- зрозумілий синтаксис;
- високий рівень абстракції;
- автоматичне керування пам'яттю;
- переносність програм;
- підтримка багатопроцесорних реалізацій.

Python також має свої недоліки такі як маленька швидкість роботи програми.

Швидкість роботи є однією з основних проблем Python та при розробці даного програмного продукту була вирішена за допомогою паралельної реалізації алгоритмів.

Для роботи з даними використовуються бібліотеки NumPy та Pandas, які дозволяють досить швидко отримувати потрібні дані.

Для розробки сервера був застосований Flask – фреймворк для розробки веб-застосунків на мові програмування Python.

Для розробки фронтенд частини використовується ReactJS, який є досить зручний і зрозумілий, що дозволяє швидко розробляти потрібні форми веб-застосунку та налаштовувати комунікацію з бекенд частиною.

					ДП 6117.00.000 ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

В якості бази даних був використаний Postgres – потужна, відкрита реляційна база даних, яка дуже швидко працює у зв’язці з Python.

4.2 Вимоги до технічного забезпечення

4.2.1 Загальні вимоги

Мінімальні вимоги до ЕОМ для коректної роботи системи:

- оперативна пам’ять – 1 Гб;
- жорсткий диск – 1 Гб;
- процесор – Intel Core i3 або краще.

Програмні засоби, що використовуються для проведення тестування:

- операційна система MacOS Catalina.

Технічні засоби, що використовуються для проведення тестування:

- оперативна пам’ять – 16 Гб;
- жорсткий диск – 500 Гб;
- процесор – Intel Core i7 2,7 ГГц;
- графічна карта - Intel Iris Plus Graphics 655.

4.3 Архітектура програмного забезпечення

4.3.1 Діаграма класів

Перелік класів та їх опис наведено у таблиці 4.1.

Таблиця 4.1 – Опис класів програмного забезпечення

Клас	Опис
PlanningDBI	Клас зберігає підключення до бази даних. Забезпечує інтерфейс для роботи з базою даних: зчитування вхідних даних, додавання вихідних результатів, зміни даних, що знаходяться у базі даних.

Продовження таблиці 4.1

Клас	Опис
DBConnect	Клас для підключення до бази даних. Зчитує дані для підключення до бази даних, обробляє їх та повертає підключення, що використовується класом PlanningDBI.
ComputingClass	Клас забезпечує обробку даних отриманих з бази даних та викликає Prediction, SalesmanComputing та BeesComputing класи для отримання результуючих даних.
Server	Клас забезпечує зв'язок з користувацьким інтерфейсом. Відповідає за запуск аналітики з відповідними налаштуваннями.
Prediction	Клас, який робить прогноз у який день потрібно інкасувати кожен з терміналів.
SalesmanComputing	Клас робить розрахунки та будує маршрут за отриманими точками маршруту для одного транспортного засобу. Повертає результуючий маршрут для одного транспортного засобу.
BeesComputing	Клас робить розрахунки та будує маршрут за отриманими точками маршруту для двох та більше транспортних засобів. Повертає результуючий маршрут для двох та більше транспортних засобів.
DataGenerating	Клас відповідає за генерацію тестових даних. Формує файли, які є вхідними даними для роботи системи.
Reporting	Клас, що відповідає за генерацію звітів, які можливо отримати з користувацького інтерфейсу.

Зв'язки між цими класами показано на діаграмі класів, що наведена у графічному матеріалі.

4.3.2 Діаграма послідовності

Розглянемо послідовність дій, що виконуються при роботі з системою.

Одразу після завантаження нового файлу сервер починає обробку файлу та отримує з нього дані, які потім завантажує до бази даних для подальшого їх використання іншими функціями.

Щоб подивитись дані, щодо наповненості терміналу, користувач повинен обрати потрібний день перегляду та термінал. Потім сервер надсилає потрібні параметри з запитом до бази даних, база даних повертає потрібні дані і сервер надсилає їх до інтерфейсу для відображення.

Для побудови нових маршрутів, користувач натискає на кнопку на інтерфейсі, сервер починає обробку даних та запускає у роботу клас, що відповідає за розрахунки, який у свою чергу запускає у роботу прогнозування, розрахунки двома методами. Потім зберігає отриманні результати у базу даних та відправляє на інтерфейс для відображення.

Для перегляду звітів користувач повинен перейти на сторінку зі звітами. Система у цей час відправляє запит до бази даних та отримує потрібні дані для побудови звітів, обробляє дані, робить розрахунки та будує графіки, які відправляються для відображення на інтерфейсі.

Послідовність виклику основних функцій програми наведена у графічному матеріалі.

4.3.3 Діаграма компонентів

Система складається з трьох основних компонентів: інтерфейсу для користувача, серверу та бази даних.

Інтерфейс відповідає за відображення даних, що зберігаються у базі даних та даних отриманих при розрахунках на сервері. Інтерфейс отримує дані з бази даних, відправляючи запити на сервер.

Сервер відповідає за розрахунки та взаємодію інтерфейсу з базою даних. Він отримує запити від інтерфейсу та обробляє їх і в свою чергу відправляє запит до бази даних через інтерфейс для спілкування з базою даних, потім відправляє відповіді на інтерфейс.

					ДП 6117.00.000 ПЗ	Арк.
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

Сервер складається з двох частин. Перша обробляє дані та надсилає їх до другої частини. Друга частина відповідає за розрахунки. Також сервер має інтерфейс для спілкування з базою даних.

База даних відповідає за зберігання даних та спілкується з сервером за допомогою запитів.

Діаграма компонентів наведена у графічному матеріалі.

4.3.4 Специфікація функцій

Опис специфікацій функцій будуть наведені для кожного класу в окремій таблиці (таблиці 4.2-4.12).

Таблиця 4.2 – Опис специфікацій функцій класу PlanningDBI

Назва функції	Вхідні дані	Вихідні дані	Опис функції
__init__	Підключення до бази даних	-	Конструктор класу, тобто метод, що виконується при створенні об'єкту класу. Отримує підключення до бази даних з класу DBConnect та зберігає це підключення для роботи іншим методам класу.
fill_cash_entities	-	-	Заповнює повністю таблицю cash_entities з файлу cash_entities.
fill_all_balances	-	-	Заповнює таблицю балансів з усіх вхідних файлів балансів.
fill_balance	date, code, percent	-	Заповнює таблицю балансів одним рядком за конкретним терміналом.
get_shipments	-	df_shipments	Виконує вибір усіх маршрутів з таблиці маршрутів.
get_coords_by_date	current_date	df_entities	Виконує вибір інтенсивностей та координат

для терміналів за обраною датою.

Продовження таблиці 4.2

Назва функції	Вхідні дані	Вихідні дані	Опис функції
fill_cost	-	-	Виконує заповнення таблиці вартостей перевезень з файлу.
get_atms	-	df_atms	Виконує вибір усіх терміналів з таблиці cash_entities.
get_entities	-	df_entities	Виконує вибір усіх терміналів та відділень банків з таблиці cash_entities.
get_balances	-	df_balances	Виконує вибір усіх балансів з таблиці балансів.
get_balanceя_by_date	date	df_balances	Виконує вибір усіх балансів за обраною датою.
add_forecast_shipment	code, next_shipment_date	-	Виконує додавання нового прогнозованого перевезення для терміналу у таблицю forecast_entity_shipment.
get_costs	-	df_costs	Виконує вибір усіх вартостей перевезень.
add_plan	date, way, length, cost	-	Виконує додавання нового маршруту до таблиці shipments.
get_ent_for_ship_by_date	date	df_entities	Виконує вибір усіх терміналів для побудови маршруту за обраною датою.
get_plan_by_date	date	df_shipment	Виконує вибір маршруту за обраною датою.
get_coords	-	df_coords	Виконує вибір координат для усіх терміналів та відділень банку.

Таблиця 4.3 – Опис специфікацій функцій класу DBConnect

Назва функції	Вхідні дані	Вихідні дані	Опис функції
get_db_conn_string	creds	connection string	Виконує побудову запиту для підключення до бази даних.
get_db_credentials	-	creds	Виконує запит на логін, пароль, назву, хост та порт підключення до бази даних.
get_db_conn_str	-	connection string	Викликає дві попередні функції для отримання логіну, паролю, назви, хоста та порту та побудови запиту до бази даних для підключення.
create_engine	connection string	connection	Виконує підключення до бази даних, яке буде отримувати PlanningDBI клас.

Таблиця 4.4 – Опис специфікацій функцій класу ComputingClass

Назва функції	Вхідні дані	Вихідні дані	Опис функції
get_ent_for_ship_by_date	date	df_entities	Виконує виклик функції, що отримує список усіх терміналів для маршруту на конкретну дату та повертає коди усіх терміналів.
return_plan	date	df_shipment	Виконує виклик побудови маршруту методами найближчого сусіда та бджолиних колоній.
get_plan_by_date	date	shipment	Виконує виклик функції, що

			повертає маршрут на обрану дату та функції. Отримання координат з бази даних та повертає список з послідовністю координат за маршрутом.
--	--	--	---

Продовження таблиці 4.4

Назва функції	Вхідні дані	Вихідні дані	Опис функції
run_analytics	-	-	Викликає функції прогнозування та розрахунків.

Таблиця 4.5 – Опис специфікацій функцій класу Server

Назва функції	Вхідні дані	Вихідні дані	Опис функції
return_intensity	date	json(cash_entities)	Приймає запит від інтерфейсу з поточною датою та викликає функцію, що робить запит у базу даних. Повертає json об'єкт з інформацією по всіх терміналах та відділення.
return_plan_by_date	date	json(shipment)	Приймає запит від інтерфейсу з поточною датою та викликає функцію, що робить запит у базу даних. Повертає json об'єкт з інформацією по всіх пунктах маршруту, що буде здійснений у поточну дату.
get_shipments	-	json(shipments)	Приймає запит від інтерфейсу та викликає

			функцію, що робить запит у базу даних. Повертає json об'єкт з інформацією по всіх маршрутах.
--	--	--	--

Продовження таблиці 4.5

Назва функції	Вхідні дані	Вихідні дані	Опис функції
run_analytics	date	-	Приймає запит від інтерфейсу з поточною датою та викликає функцію, що робить прогнозування та побудову маршрутів та зберігає їх у базу даних.
upload_file			Завантажує отриманні з інтерфейсу файли.

Таблиця 4.6 – Опис специфікацій функцій класу Prediction

Назва функції	Вхідні дані	Вихідні дані	Опис функції
day_for_enity			Прогнозує день для перевезення за терміналом. Та викликає функцію додавання нового перевезення для терміналу до бази даних.

Таблиця 4.7 – Опис специфікацій функцій класу SalesmanComputing

Назва функції	Вхідні дані	Вихідні дані	Опис функції
salesman_best_way			Виконує паралельний запуск розрахунків методом

			найближчого сусіда та повертає найкращий маршрут.
find_way			Виконує алгоритм методу найближчого сусіда.

Таблиця 4.8 – Опис специфікацій функцій класу BeesComputing

Назва функції	Вхідні дані	Вихідні дані	Опис функції
GenerateWay			Виконує генерацію нового випадкового маршруту.
fitness_func			Виконує розрахунок фітнес-функції для обраного маршруту.
GenerateNeighborWay			Виконує генерацію сусідського випадкового маршруту.
Hive()			Ініціалізує вулик з бджолами.

Таблиця 4.9 – Опис специфікацій функцій класу Hive

Назва функції	Вхідні дані	Вихідні дані	Опис функції
__init__			Конструктор класу Hive. Виконує ініціалізацію бджолиного вулика та генерацію популяції бджіл.
solve			Виконує запуск алгоритму методом бджолиних колоній.
Bee()			Ініціалізує бджолу.

Таблиця 4.10 – Опис специфікацій функцій класу Bee

Назва функції	Вхідні	Вихідні	Опис функції
---------------	--------	---------	--------------

	дані	дані	
__init__			Конструктор класу Bee. Виконує ініціалізацію бджоли та маршруту для бджоли.
bee_sort			Виконує перевантаження функції сортування для об'єкта бджоли.

Таблиця 4.11 – Опис специфікацій функцій класу DataGenerating

Назва функції	Вхідні дані	Вихідні дані	Опис функції
gen_cash_entities_file			Виконує генерацію терміналів та відділень банку та зв'язків між ними та генерує файл cash_entities як вхідний файл для тестування програми.
gen_balances_files			Виконує генерацію файлів балансів на відповідний проміжок часу, задаючи випадкові інтенсивності наповненості терміналів та випадкові відхилення для імітації наповненості терміналів.
gen_durations_file			Виконує підключення до GoogleMaps API для отримання дистанцій між терміналами. Виконує запуск функції для отримання списку усіх терміналів та формує файл дистанцій та вартостей як вхідного файлу для тестування програми.

Таблиця 4.12 – Опис специфікацій функцій класу Reporting

					ДП 6117.00.000 ПЗ	Арк.
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

Назва функції	Вхідні дані	Вихідні дані	Опис функції
cost_by_days	-	json	Викликає функції, що роблять запити у базу даних для отримання інформації по вартостях маршрутів по кожному перевезенню, обробляє дані та повертає json об'єкт для побудови графіків.

Продовження таблиці 4.12

Назва функції	Вхідні дані	Вихідні дані	Опис функції
count_by_days	-	json	Викликає функції, що роблять запити у базу даних для отримання інформації по кількостях інкасованих терміналів по кожному дню, обробляє дані та повертає json об'єкт для побудови графіків.
compare_cost	-	json	Викликає функції, що роблять запити у базу даних для отримання інформації по вартостях маршрутів по кожному перевезенню, обробляє дані, порівнює їх з теоретичними даними для періодичних перевозок та повертає json об'єкт для побудови графіків.
intensity_entity	entity_code	json	Викликає функції, що роблять запити у базу даних для отримання інформації наповненості терміналу по

Змн.	Арк.	№ докум.	Підпис	Дата

		кожному дню, обробляє дані та повертає json об'єкт для побудови графіків.
--	--	---

4.4 Опис звітів

Веб-застосунок складається з 3 сторінок, на яких міститься весь функціонал:

- головна сторінка (наведено на рисунку 4.1);
- сторінка завантаження файлів (наведено на рисунку 4.2);
- сторінка зі звітами (наведено на рисунку 4.3).

Головна сторінка відповідає за весь основний функціонал: запуск побудови нових маршрутів, перегляд наповненості терміналів, перегляд усіх маршрутів, що були побудовані.

Приклад побудованого маршруту наведено на рисунку 4.5 та приклад головної сторінки з побудованим маршрутом наведено на рисунку 4.4.

Приклад перегляду наповненості терміналу наведено на рисунку 4.6.

Bank Analytics

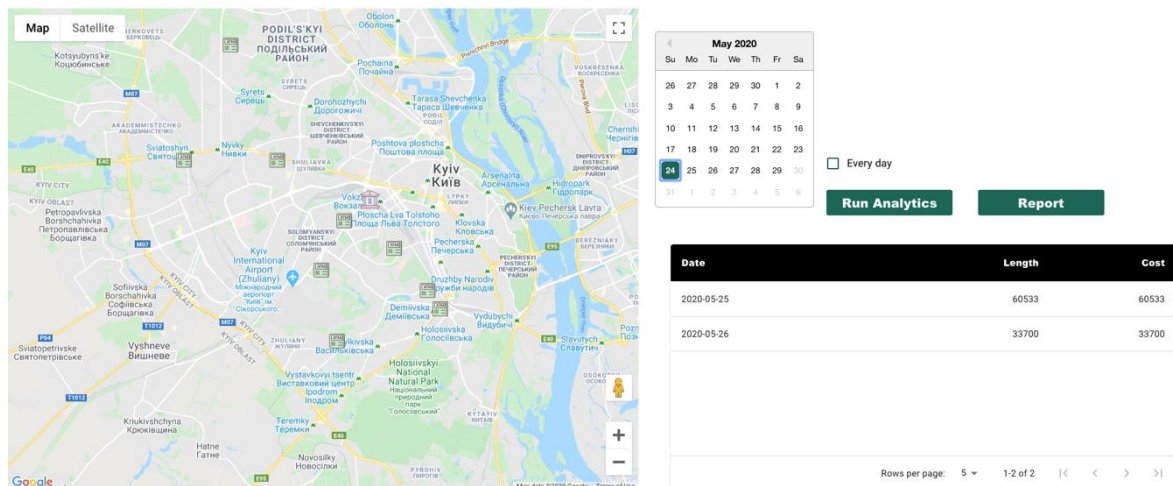


Рисунок 4.1 – Приклад головної сторінки на сьогоднішній день

Bank Analytics

Drag'n/drop files, or click to select files

Files:

Рисунок 4.2 – Сторінка для завантаження файлів

Bank Analytics

Upload Files

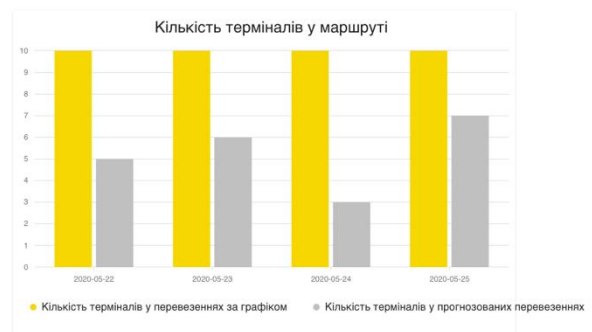
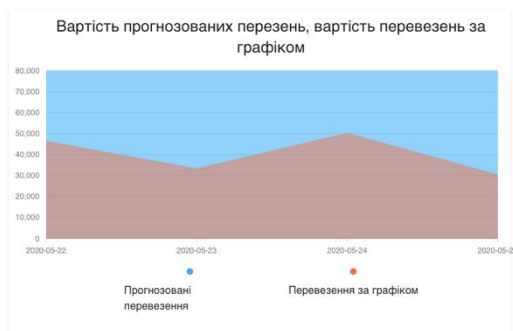
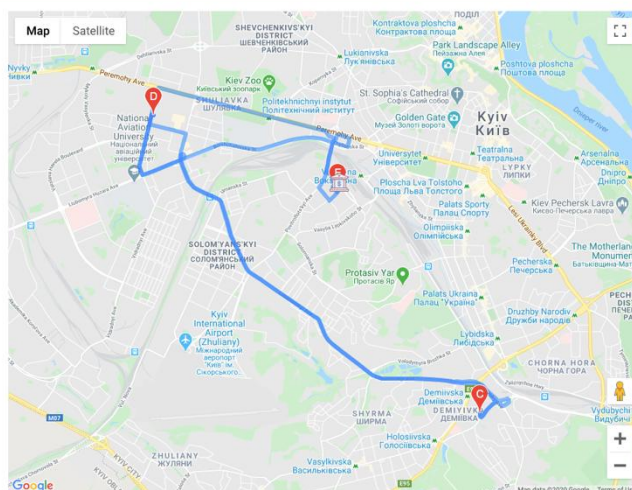


Рисунок 4.3 – Сторінка зі звітами

Bank Analytics



May 2020

Su	Mo	Tu	We	Th	Fr	Sa
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

☐ Every day

Report

Date	Length	Cost
2020-05-25	60533	60533
2020-05-26	33700	33700

Rows per page: 5 1-2 of 2

Рисунок 4.4 – Приклад головної сторінки на день з наявними маршрутами

					ДП 6117.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		53

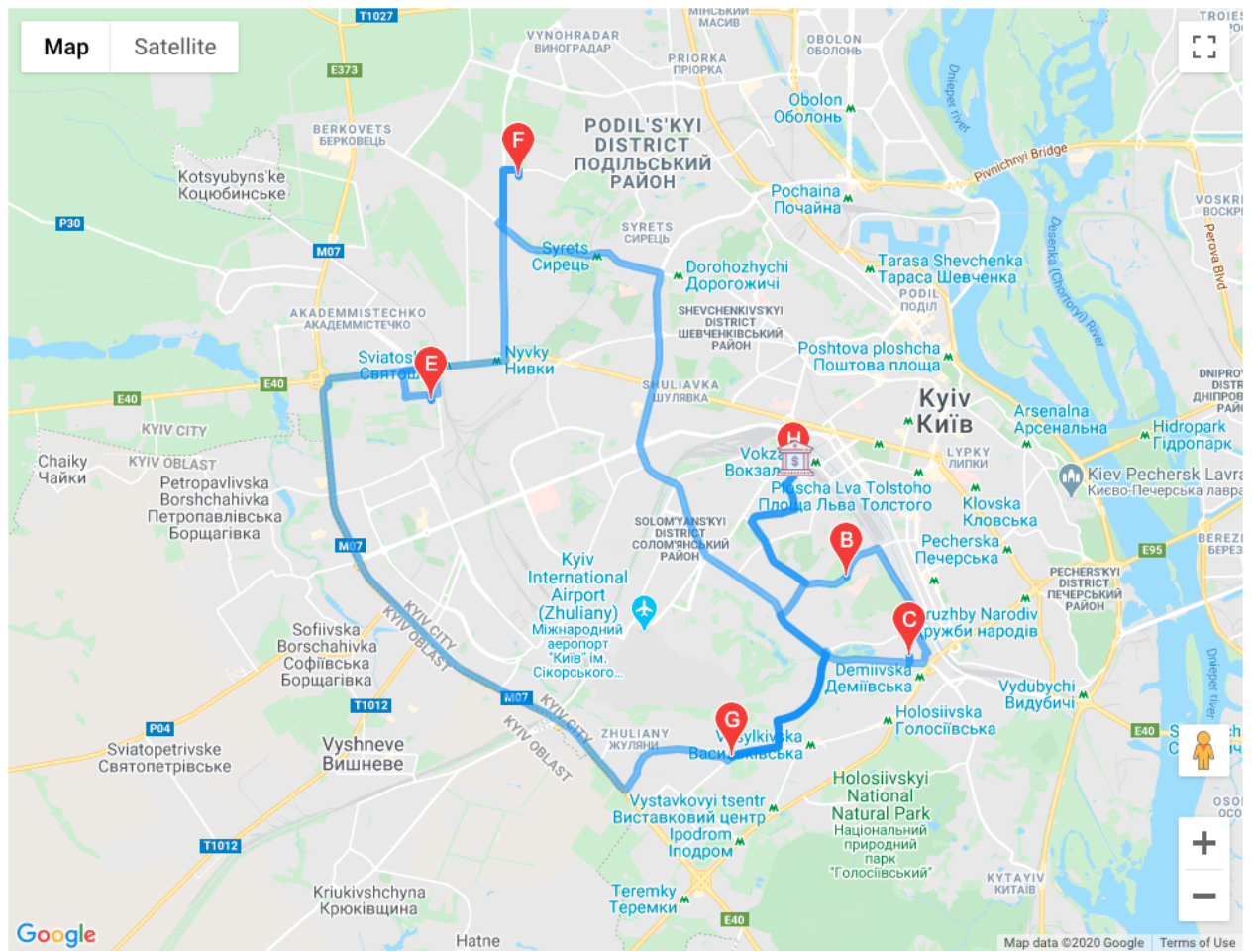


Рисунок 4.5 – Приклад мапи з побудованим маршрутом

					ДП 6117.00.000 ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		54

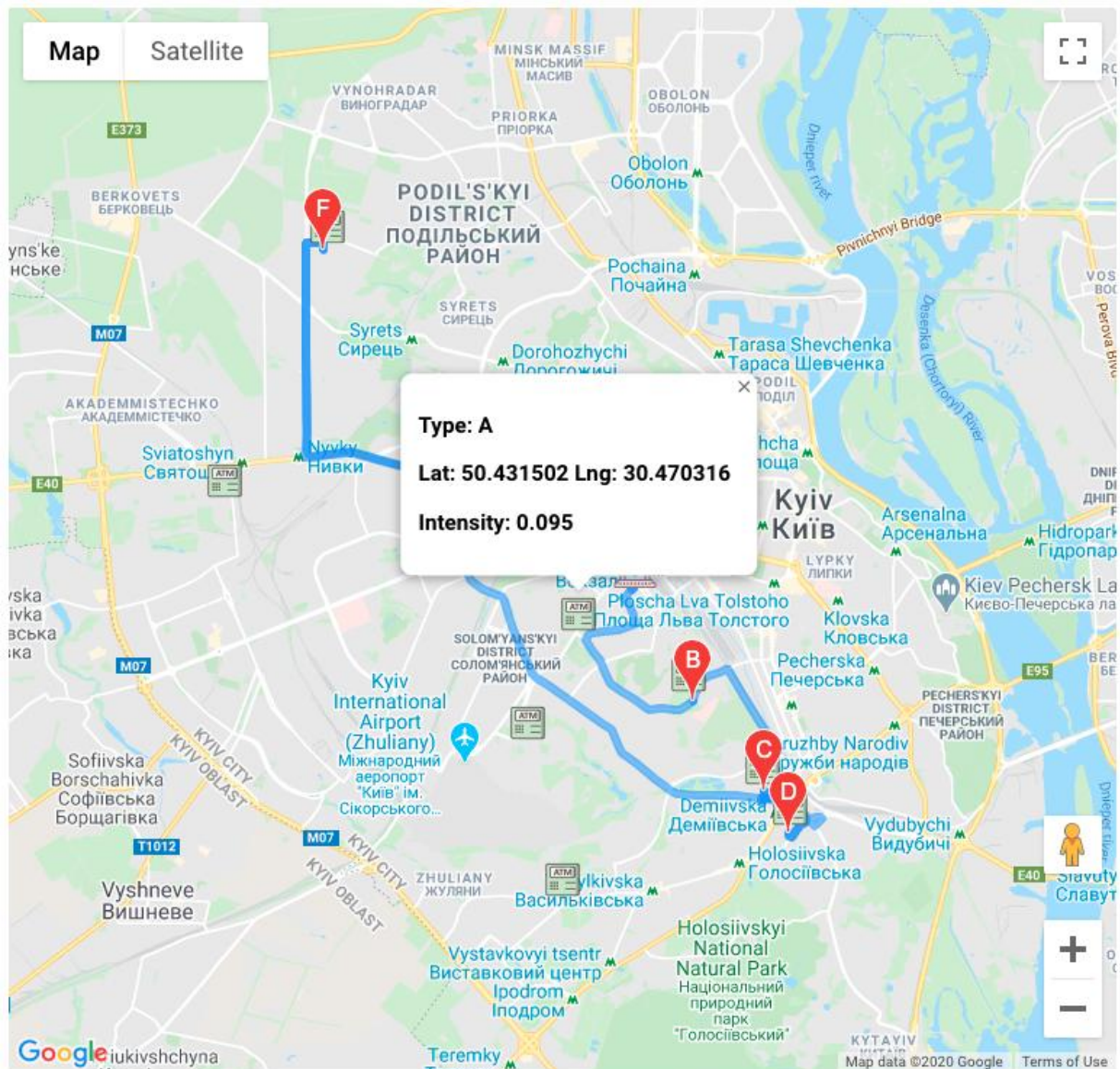


Рисунок 4.6 – Приклад інформаційного вікна для терміналу

Висновок до розділу

У розділі обґрунтовано обрані засоби розробки: Python, ReactJS, Postgres. Для кожної технології розглянуто переваги та недоліки у розрізі конкретного проекту. У розділі також наведено діаграму класів та опис кожного класу, а також детальний опис усіх функцій програми. Наведено діаграми послідовності та діаграми компонентів даного проекту.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

5.1.1 Завантаження файлів

Для початку використання застосунку потрібно завантажити усі необхідні файли. Серед обов'язкових файлів: файл усіх терміналів та відділень (cash_entities.csv), файли балансів за попередні дати (balances.<date>.csv) та файл відстаней між терміналами (distances.csv).

Для завантаження файлів необхідно натиснути кнопку “Upload Files”, що знаходиться у верхньому правому куті головної сторінки (рисунок 5.1).

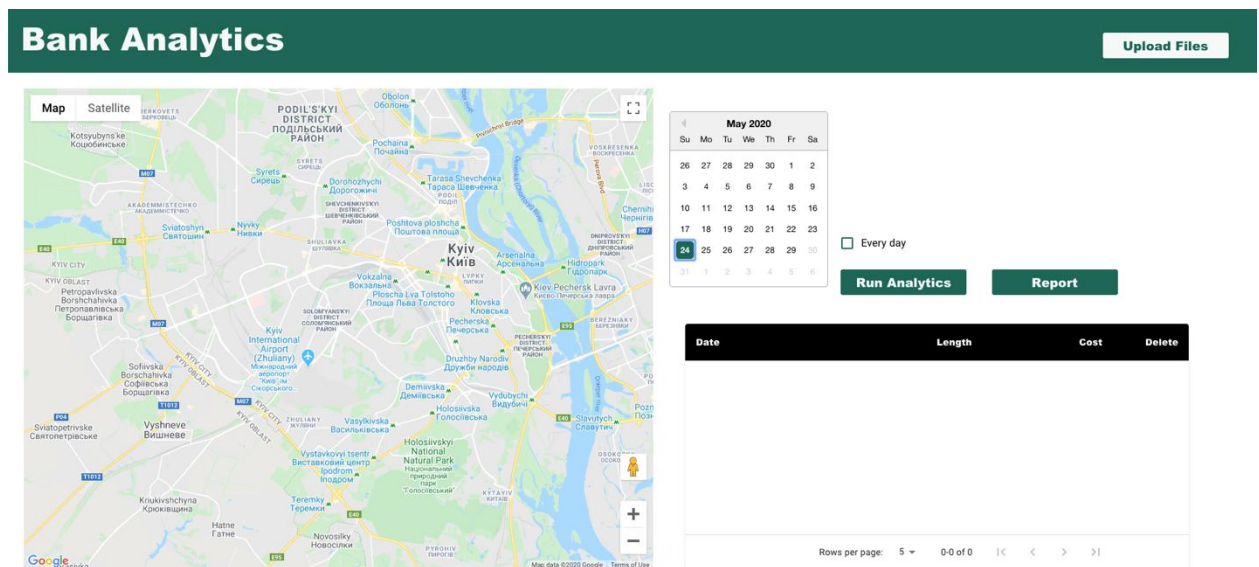


Рисунок 5.1 – Головна сторінка застосунку без завантажених файлів

Після натискання кнопки “Upload Files” з’явиться сторінка для завантаження файлів (рисунок 5.2). Для завантаження потрібно перетягнути файл або натиснути на зону завантаження та обрати потрібний файл у директорії (рисунок 5.3). Після завантаження файлів під зоною завантаження з’явиться список файлів, що були завантажені (рисунок 5.4).

Змн.	Арк.	№ докум.	Підпис	Дата

Bank Analytics



Рисунок 5.2 – Сторінка для завантаження файлів у початковому стані

Bank Analytics



Рисунок 5.3 – Сторінка для завантаження файлів у момент завантаження файлу

Bank Analytics

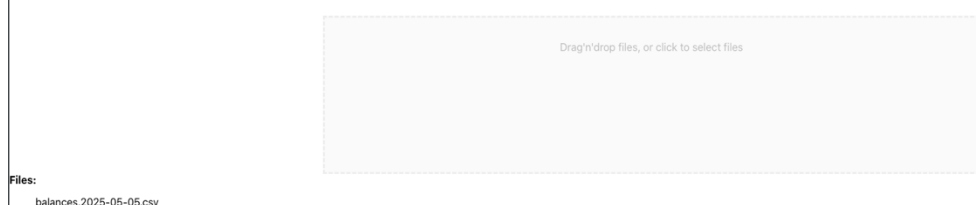


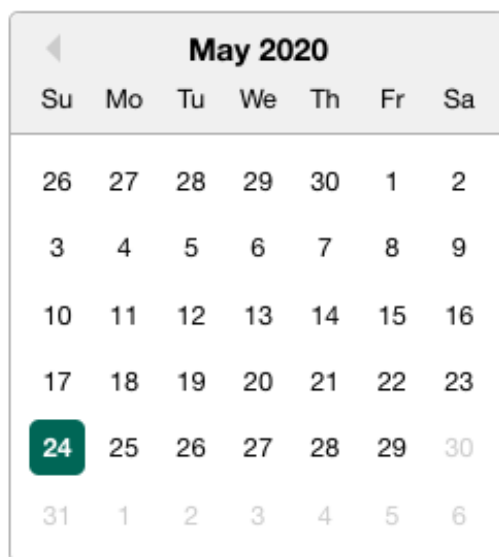
Рисунок 5.4 – Сторінка для завантаження файлів після завантаження файлу

5.1.2 Перегляд балансів терміналів

Після відкриття веб-застосунку з уже завантаженими даними можна одразу приступати до роботи. Для перегляду історичних балансів терміналів

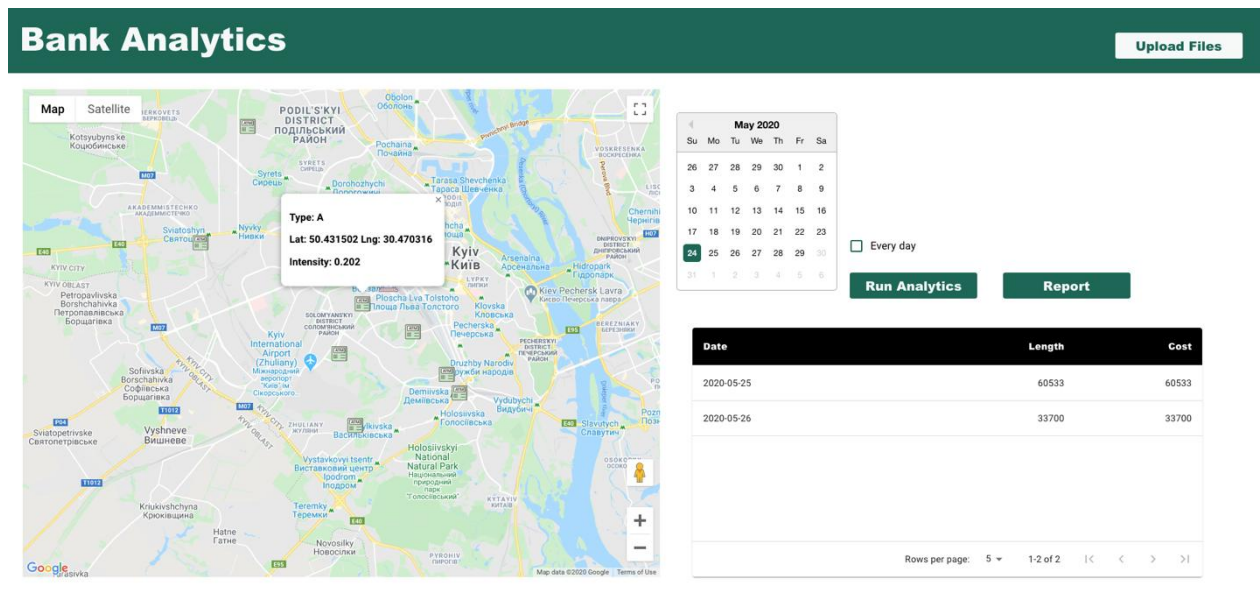
					ДП 6117.00.000 ПЗ	Арк.
						57
Змн.	Арк.	№ докум.	Підпис	Дата		

можна обирати потрібну дату на календарі та на мапі буде відображатися поточний стан за конкретною датою (рисуюнок 5.5).



Рисуюнок 5.5 – Календар для обрання потрібної дати перегляду

Після обрання потрібної дати на мапі можна обрати термінал та натиснути на нього для перегляду інформації (рисуюнок 5.6).



Рисуюнок 5.6 – Перегляд інформації по терміналу

5.1.3 Побудова маршруту

Якщо у базі даних ще немає ніяких маршрутів, то і в таблиці маршрутів ще не буде ні одного маршруту.

Для побудови маршрутів можна обрати одну з двох опцій: побудувати маршрути на кожен день на найближчі два дні (якщо такі існують) або побудувати маршрут через день (якщо такі існують).

Для побудови маршрутів на кожен день потрібно натиснути на прапорець, як на рисунку 5.7. Або залишити ненатиснутим, як на рисунку 5.8.

May 2020						
Su	Mo	Tu	We	Th	Fr	Sa
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

☒ Every day

Run Analytics

Report

Рисунок 5.7 – Приклад натиснутого прапорця для побудови маршрутів на кожен день

May 2020						
Su	Mo	Tu	We	Th	Fr	Sa
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

☐ Every day

Run Analytics

Report

Рисунок 5.8 – Приклад ненатиснутого прапорця для побудови маршрутів через день

Змн.	Арк.	№ докум.	Підпис	Дата

Після натискання кнопки “Run Analytics”, будуть побудовані можливі маршрути на наступні два дні. І результати можна буде побачити у таблиці маршрутів (рисунок 5.9).



Date	Length	Cost	Delete
2020-05-26	60533	60533	
2020-05-27	33700	33700	
Rows per page: 5 ▾ 1-2 of 2 < < > >			

Рисунок 5.9 – Таблиця маршрутів

Подивитись маршрут можна двома способами: натискаючи на відповідну дату на календарі або натискаючи на відповідний маршрут у таблиці маршрутів. Обравши потрібний день одним з двох варіантів, на мапі буде відображено відповідний маршрут (рисунок 5.10).

Bank Analytics

Upload Files

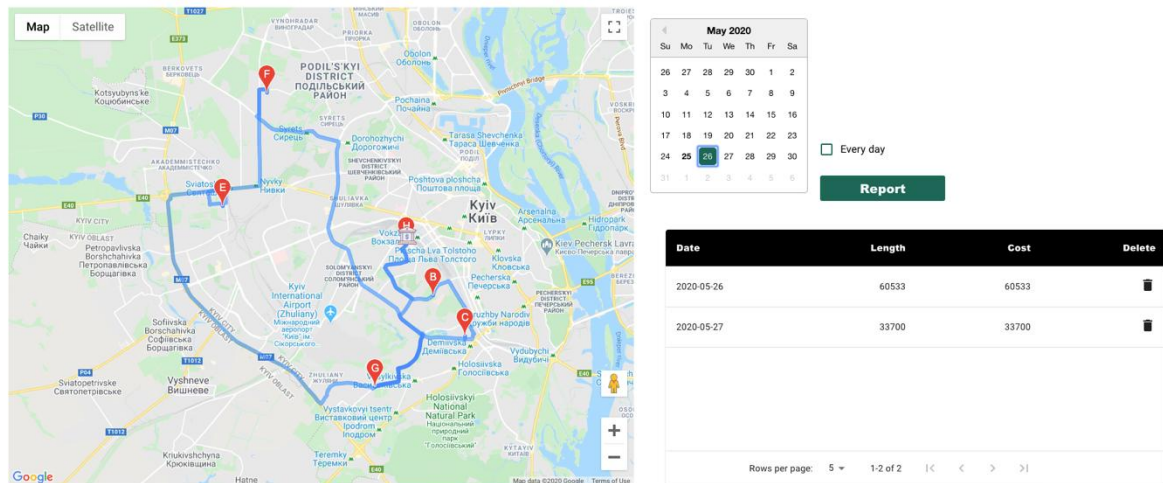


Рисунок 5.10 – Головна сторінка з побудованим маршрутом

Використовуючи кнопку видалення, можна видаляти непотрібні маршрути.

5.1.4 Перегляд звітів

Натиснувши на кнопку “Report” на головній сторінці можна потрапити на сторінку звітів на переглянути згенеровані графіки по даним що були отримані під час роботи програми (рисунок 5.11).

Bank Analytics

Upload Files



Рисунок 5.11 – Сторінка зі звітами

Змн.	Арк.	№ докум.	Підпис	Дата

5.2 Випробування програмного продукту

В цьому підрозділі наведено опис тестів і порядок їх виконання для перевірки відповідності програмного забезпечення комплексу задач функціональним вимогам, представленим у технічному завданні на створення комплексу задач складання планів інкасації терміналів з прийому платежів.

5.2.1 Мета випробувань

Метою випробувань є перевірка відповідності функцій комплексу задач складання планів інкасації терміналів з прийому платежів вимогам технічного завдання.

5.2.2 Загальні положення

Випробування проводяться на основі наступних документів:

- ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
- ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

Було проведено тестування усієї функціональності комплексу задач. У таблицях 5.1 – 5.5 наведено випробування основних функцій.

Таблиця 5.1 – Тестування завантаження файлів

Мета тесту	Перевірка функції завантаження файлів
Початковий стан	Відкрита головна сторінка системи
Вхідні дані	Файл для завантаження

Продовження таблиці 5.1

Схема проведення тесту	<ol style="list-style-type: none"> 1. Натиснути на кнопку “Upload Files”; 2. Завантажити файл: <ul style="list-style-type: none"> - перетягнути потрібний файл у поле для завантаження; - натиснути на поле для завантаження та обрати потрібний файл з директорії.
Очікуваний результат	Назва завантаженого файлу з’являється у списку завантажених файлів
Стан системи після проведення випробувань	Назва завантаженого файлу з’явилась у списку завантажених файлів

Таблиця 5.2 – Тестування функції перегляду завантаженості терміналу

Мета тесту	Перевірка функції перегляду завантаженості терміналу
Початковий стан	Відкрита головна сторінка системи
Вхідні дані	Дата, термінал
Схема проведення тесту	<ol style="list-style-type: none"> 1. Обрати потрібний день на календарі 2. Обрати потрібний термінал на мапі
Очікуваний результат	З’являється інформаційне вікно з інформацією про термінал на мапі
Стан системи після проведення випробувань	З’явилося інформаційне вікно з інформацією про термінал на мапі

Таблиця 5.3 – Тестування функції побудови маршруту

Мета тесту	Перевірка функції побудови маршруту
Початковий стан	Відкрита головна сторінка системи
Вхідні дані	-
Схема проведення тесту	<ol style="list-style-type: none"> 1. Обрати сьогоднішній день 2. Натиснути кнопку “Run Analytics”
Очікуваний результат	З’являється повідомлення щодо успішного запуску побудови маршрутів. У таблиці маршрутів протягом 5 хвилин з’являються нові побудовані маршрути

Продовження таблиці 5.3

Стан системи після проведення випробувань	З'явилося повідомлення щодо успішного запуску побудови маршрутів. У таблиці маршрутів через 2 хвилини з'явилися нові побудовані маршрути
---	--

Таблиця 5.4 – Тестування функції перегляду звітів

Мета тесту	Перевірка функції перегляду звітів
Початковий стан	Відкрита головна сторінка системи
Вхідні дані	-
Схема проведення тесту	Натиснути кнопку "Report"
Очікуваний результат	Відкрита сторінка зі звітами
Стан системи після проведення випробувань	Відкрита сторінка зі звітами

Таблиця 5.5 – Тестування функції видалення маршрутів

Мета тесту	Перевірка функції видалення маршрутів
Початковий стан	Відкрита головна сторінка системи
Вхідні дані	-
Схема проведення тесту	1. Натиснути кнопку "Delete" у таблиці маршрутів. 2. Підтвердити видалення
Очікуваний результат	Маршрут видалено за таблиці маршрутів
Стан системи після проведення випробувань	Маршрут видалено за таблиці маршрутів

Висновок до розділу

У розділі наведено інструкцію користувача з описаними діями, які можна виконувати, описана послідовність роботи в залежності від дій. Також наведено випробування системи, набори тестів, що дозволяють перевірити правильність роботи системи.

ЗАГАЛЬНІ ВИСНОВКИ

Під час роботи над дипломним проектом були розглянуті основні етапи життєвого циклу програмного продукту. Були виділені основні етапи роботи над програмним продуктом, проаналізовано предметне середовище, описані бізнес-процеси та бізнес-задачі, розроблено процес діяльності, за яким повинна працювати система, визначені функції користувачів та розроблена діаграма діяльності. Описано призначення, мета та задачі, які потрібно виконати для досягнення мети.

У розділі інформаційного забезпечення були визначенні вхідні та вихідні дані та документи системи, визначено таблиці бази даних та побудована схема бази даних.

У розділі математичного забезпечення було представлено постановку задачі маршрутизації транспортних засобів VRP та методи її розв'язання для двох випадків. Для методів було створено узагальнені схеми алгоритмів. Було застосовано модифікації класичних алгоритмів для можливості їх паралельної реалізації, що дозволить швидше отримувати результати, зберігаючи точність розрахунків, або за той же час отримувати кращі результати.

У розділі програмного забезпечення обґрунтовано обрані засоби розробки: Python, ReactJS, Postgres. Для кожної технології розглянуто переваги та недоліки у розрізі конкретного проекту. У розділі також наведено діаграму класів та опис кожного класу, а також детальний опис усіх функцій програми. Наведено діаграми послідовності та діаграми компонентів даного проекту.

У технологічному розділі наведено інструкцію користувача з описаними діями, які можна виконувати, описана послідовність роботи в залежності від дій. Також наведено випробування системи, набори тестів, що дозволяють перевірити правильність роботи системи.

					ДП 6117.00.000 ПЗ	Арк.
						65
Змн.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК ПОСИЛАНЬ

1. Рассадникова Е.Ю., Коханчиков Л.А. Математическая модель задачи выбора рациональных маршрутов в системе управления транспортировки готовой продукции // Современные проблемы науки и образования. – 2013. – № 5.;
2. Лукова О.Ю. Задача складання планів інкасацій терміналів з прийому платежів у населення / Лукова О.Ю., Жданова О.Г. // Матеріали IV всеукраїнської науково-практичної конференції молодих вчених та студентів «Інформаційні системи та технології управління» (ІСТУ-2020) – м. Київ.: НТУУ «КПІ ім. Ігоря Сікорського», 30 квітня 2020 р. – С. 94-98;
3. Гришин А. А., Карпенко А. П. Исследование эффективности метода пчелиного роя в задаче глобальной оптимизации // Машиностроение и компьютерные технологии. 2010. №08. URL: <https://cyberleninka.ru/article/n/issledovanie-effektivnosti-metoda-pchelinogo-roya-v-zadache-globalnoy-optimizatsii>

Додаток А

Тексти програмного коду

Система складання планів обслуговування терміналів з прийому платежів у населення.

(Найменування програми (документа))

DVD-R

(Вид носія даних)

20 арк,

(Обсяг програми (документа) , арк., Кб)

Київ – 2020 року

					ДП 6117.00.000 ПЗ	Арк.
						67
Змн.	Арк.	№ докум.	Підпис	Дата		


```

from flask import Flask, request, jsonify
from flask_cors import CORS
import numpy as np
from sqlalchemy import create_engine
import pandas as pd

from backend.app.computing import run_analytics_for_params, get_plan_by_date
from db.dbi import DiplomDB, get_db_connection_str

app = Flask(__name__)
cors = CORS(app, resources={r"/*": {"origins": "*"}})

engine = create_engine(get_db_connection_str())

db = DiplomDB(engine)

@app.route('/')
@app.route('/index')
def index():
    return "Hello, World!"

# получаем дату и возвращаем текущие интенсивности
@app.route('/intensity')
def return_intensity():
    date = request.args.get('date')
    current_date = pd.to_datetime(date).strftime("%Y-%m-%d")
    df = db.return_intensities_and_coords_by_date(current_date)
    df_branch = db.return_branch()
    cash_entities = []

    for index, row in df_branch.iterrows():
        cash_entities.append({
            'type': row['ce_type'],
            'lat': row['x_coord'],
            'lng': row['y_coord'],
            'intensity': 1
        })

    for index, row in df.iterrows():
        cash_entities.append({
            'type': row['ce_type'],
            'lat': row['x_coord'],
            'lng': row['y_coord'],
            'intensity': row['percent']
        })

    print(cash_entities)

    return jsonify({
        'cash_entities': cash_entities
    })

```

получаем дату и выводим план на этот день, иначе пустой массив

					ДП 6117.00.000 ПЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

```
@app.route('/plan_by_date')
def return_plan_by_date():
    date = request.args.get('date')
    print(get_plan_by_date(date))
    return jsonify({
        'plan': get_plan_by_date(date)
    })
```

вивод списка с бд

```
@app.route('/shipments')
def get_shipments():
    df = db.return_shipments()
    shipments = []

    for index, row in df.iterrows():
        shipments.append({
            'date': row['date'].strftime("%Y-%m-%d"),
            'length': row['length'],
            'cost': row['cost']
        })

    return jsonify({
        'shipments': shipments
    })
```

```
@app.route('/run_analytics')
def run_analytics():
    date = request.args.get('date')
    #frequency = request.args.get('frequency')

    current_date = pd.to_datetime(date).strftime("%Y-%m-%d")

    try:
        run_analytics_for_params(current_date)
        return 'Success', 200
    except:
        print('error')
        return 'Error', 500
```

```
if __name__ == '__main__':
    np.random.seed(4)
    app.run(threaded=False, debug=True, host='0.0.0.0', port='5002')
```

```
from sqlalchemy import create_engine
import pandas as pd
import numpy as np
from pathlib import Path
```

```
def get_db_connection_string(creds):
    return
f'postgres://{creds["user"]}::{creds["password"]}@{creds["host"]}::{creds["port"]}/{creds["data"]}
```

					ДП 6117.00.000 ПЗ	Арк.
						69
Змн.	Арк.	№ докум.	Підпис	Дата		

```
base"]}]'
```

```
def get_db_credentials():
    return {
        "host": 'localhost',
        "port": '5433',
        "database": 'postgres',
        "user": 'postgres',
        "password": 'postgres'
    }
```

```
def get_db_connection_str():
    creds = get_db_credentials()
    return get_db_connection_string(creds)
```

```
class DiplomDB:
    def __init__(
        self,
        connectable
    ):
        """
        Use SQLAlchemy Connectable to perform ACE queries.
        """
        self.connectable = connectable
```

```
def fill_cash_entities(self):
    df = pd.read_csv('./data/cash_entities/cash_entities.csv',
                     dtype={
                         'X_COORD': np.str,
                         'Y_COORD': np.str
                     })
    with self.connectable.connect() as conn:
        for index, row in df.iterrows():
            sql = (
                f"""
                INSERT INTO cash_entity (
                    CE_CODE,
                    CE_NAME,
                    CE_TYPE,
                    PARENT_CODE,
                    X_COORD,
                    Y_COORD
                ) VALUES (
                    {row['CODE']},
                    '{row['NAME']}',
                    '{row['TYPE']}',
                    {row['PARENT_CODE']},

                    '{row['X_COORD']}',
                    '{row['Y_COORD']}'
                )"""
            )
            conn.execute(sql)
```

```

def fill_all_balances(self):
    path = Path("../data/balances")

    df = (
        pd.concat([
            pd.read_csv(f, dtype={'PERCENT': np.str})
            for f in path.glob("*.csv")
        ]).sort_values(by=['DATE', 'CODE'])
    )

    with self.connectable.connect() as conn:
        for index, row in df.iterrows():
            sql = (
                f"""
                INSERT INTO balance (date, code, percent)
                VALUES (
                    '{row['DATE']}',
                    '{row['CODE']}',
                    '{row['PERCENT']}'
                )"""
            )
            conn.execute(sql)

    def fill_balance(self, date, code, percent):
        with self.connectable.connect() as conn:
            sql = (
                f"""
                INSERT INTO forecast_balance (date, code, percent)
                VALUES (
                    '{date}',
                    '{code}',
                    '{percent}'
                )"""
            )
            conn.execute(sql)

    def return_shipments(self):
        sql = f'select * from shipments'

        df = pd.read_sql(sql, self.connectable)

        return df

    def return_intensities_and_coords_by_date(self, current_date):
        sql = f"""
        select ce_type, x_coord, y_coord, percent
        from cash_entity

        join balance b on cash_entity.ce_code = b.code
        where date= '{current_date}'
        """

        df = pd.read_sql(sql, self.connectable)

        return df

```

```

def return_branch(self):
    sql = f'''
        select ce_type, x_coord, y_coord
        from cash_entity
        where ce_code=1
    '''

    df = pd.read_sql(sql, self.connectable)

    return df

def fill_cost(self):
    df = pd.read_csv('./data/distances/distances.csv')
    with self.connectable.connect() as conn:
        for index, row in df.iterrows():
            sql = (
                f'''
                    INSERT INTO cost (
                        FROM_CODE,
                        TO_CODE,
                        DISTANCE,
                        DURATION
                    ) VALUES (
                        {row['FROM_CODE']},
                        {row['TO_CODE']},
                        {row['DISTANCE']},
                        {row['DURATION']}
                    )'''
            )
            conn.execute(sql)

def get_min_max_date(self):
    sql = f'''
        select min(date), max(date)
        from balance
    '''

    df = pd.read_sql(sql, self.connectable)

    return df

def get_atms(self):
    sql = f'''
        select ce_code
        from cash_entity

        where ce_type='A'
    '''

    df = pd.read_sql(sql, self.connectable)

    return df

def get_entities(self, depo):
    sql = f'''

```

```

select *
from cash_entity
where parent_code = {depo}
...

```

```
df = pd.read_sql(sql, self.connectable)
```

```
return df
```

```
def get_balances(self):
```

```
    sql = f'''
```

```

        select *
        from balance
    ...

```

```
df = pd.read_sql(sql, self.connectable)
```

```
return df
```

```
def get_balances_by_date(self, date):
```

```
    sql = f'''
```

```

        select *
        from balance
        where date='{date}'
    ...

```

```
df = pd.read_sql(sql, self.connectable)
```

```
return df
```

```
def add_forecast_shipment(self,
```

```
    ce_code,
```

```
    next_shipment,
```

```
):
```

```
with self.connectable.connect() as conn:
```

```
    sql = (
```

```
        f'''
```

```

            INSERT INTO forecast_entity_shipment (

```

```
                CE_CODE,
```

```
                NEXT_DATE
```

```
            ) VALUES (
```

```
                {ce_code},
```

```
                '{next_shipment}'
```

```
            )'''
```

```
        )
```

```
    )
```

```
    conn.execute(sql)
```

```
def get_costs(self):
```

```
    sql = f'''
```

```

        select *

```

```
        from cost
```

```
    ...

```

```
df = pd.read_sql(sql, self.connectable)
```

```

return df

def add_plan(self, date, way, length, cost):
    with self.connectable.connect() as conn:
        sql = (
            f'''
            INSERT INTO shipments (
                DATE,
                POINT,
                LENGTH,
                COST
            ) VALUES (
                '{date}',
                ARRAY {way},
                {length},
                {cost}
            )'''
        )
        conn.execute(sql)

def get_all_next_shipment_dates(self):
    sql = f'''
    select distinct first_next_date
    from forecast_entity_shipment
    '''

    df = pd.read_sql(sql, self.connectable)

    return df

def get_entities_for_shipment_by_date(self, date):
    sql = f'''
    select ce_code
    from forecast_entity_shipment
    where next_date='{date}'
    '''

    df = pd.read_sql(sql, self.connectable)

    return df

def get_plan_by_date(self, date):
    sql = f'''
    select point
    from shipments
    where date='{date}'
    '''

    df = pd.read_sql(sql, self.connectable)

    return df

def get_coords(self):
    sql = f'''

```

Змн.	Арк.	№ докум.	Підпис	Дата

```
select ce_code, x_coord, y_coord
from cash_entity
...
```

```
df = pd.read_sql(sql, self.connectable)
```

```
return df
```

```
engine = create_engine(get_db_connection_str())
```

```
db = DiplomDB(engine)
```

```
import random
```

```
import numpy as np
```

```
from numpy import exp, sqrt
```

```
import matplotlib.pyplot as plt
```

```
import time
```

```
from concurrent.futures import ProcessPoolExecutor, wait
```

```
import copy
```

```
def find_way(ib, X, M, depo):
```

```
    costs = copy.deepcopy(M)
```

```
    way = [depo, ib]
```

```
    S = 0
```

```
    for i in X:
```

```
        M[i][depo] = float('inf')
```

```
        M[i][ib] = float('inf')
```

```
    i = way[-1]
```

```
    for _ in range(len(X)-1):
```

```
        i = way[-1]
```

```
        min_index = min(M[i], key=M[i].get)
```

```
        way.append(min_index)
```

```
    for j in range(len(X)):
```

```
        M[X[j]][min_index] = float('inf')
```

```
    way.append(depo)
```

```
    for i in range(len(way)-1):
```

```
        S += costs[way[i]][way[i+1]]
```

```
    return S, way, ib
```

```
def X_sort(elem):
```

```
    return M[elem][depo]
```

```
def best_way(M, X, depo):
```

```
    RS = []
```

					ДП 6117.00.000 ПЗ	Арк.
						75
Змн.	Арк.	№ докум.	Підпис	Дата		


```

RW = []
RIB = []
s = []

X.sort(key=X_sort)

with ProcessPoolExecutor(6) as executor:
    tasks = [
        executor.submit(find_way, ib=ib, X=X, M=M, depo=depo)
        for ib in X[:len(X)//2]
    ]
    # we must wait until the whole layer processing finished:
    wait(tasks)

for i in range(len(tasks)):
    RS.append(tasks[i].result()[0])
    RW.append(tasks[i].result()[1])
    RIB.append(tasks[i].result()[2])

S = min(RS)
result_way = RW[RS.index(min(RS))]
ib = RIB[RS.index(min(RS))]

return S, ib, result_way

n = 10
m = 100
a = 0
x = np.random.uniform(a, m, n)
y = np.random.uniform(a, m, n)

M = {}
for i in np.arange(0, n, 1):
    M[i] = {}
    for j in np.arange(0, n, 1):
        if i != j:
            M[i][j] = sqrt((x[i] - x[j]) ** 2 + (y[i] - y[j]) ** 2)

        else:
            M[i][j] = float('inf')

X = [i for i in range(1, n)]

depo = 0
quatilies = []

scout_bee_count = 20
selected_bee_count = 4
best_bee_count = 5
sel_sites_count = 5
best_sites_count = 4
population = []

def GenerateWay(X, depo):
    result = [depo]

```

					ДП 6117.00.000 ПЗ	Арк.
						76
Змн.	Арк.	№ докум.	Підпис	Дата		

```

random.shuffle(X)
result += X
result.append(depo)
return result

def GenerateNeighborWay(way):
    way1 = way.copy()
    for i in range(3):
        maximum = 0
        point = 0
        for i in range(1, len(way1)-1):
            if M[way1[i]][way1[i+1]] > maximum:
                maximum = M[way1[i]][way1[i+1]]
                point = i

        first_index = point
        second_index = random.randrange(1, len(way1)-1)
        way1[first_index], way1[second_index] = way1[second_index], way1[first_index]
    return way1

def fitness_func(way):
    S = 0
    for i in range(len(way) - 1):
        S += M[way[i]][way[i + 1]]
    return S

class Hive:
    def __init__(self,
        scout_bee_count,
        selected_bee_count,
        best_bee_count,
        sel_sites_count,
        best_sites_count
    ):
        self.scout_bee_count = scout_bee_count

        self.selected_bee_count = selected_bee_count
        self.best_bee_count = best_bee_count

        self.sel_sites_count = sel_sites_count
        self.best_sites_count = best_sites_count

        self.best_sites = []
        self.sel_sites = []
        self.beecount = scout_bee_count + selected_bee_count * sel_sites_count + best_bee_count *
        best_sites_count

        self.swarm = [Bee() for _ in range(self.beecount)]
        self.swarm.sort(key=Bee.bee_sort)

        self.best_position = self.swarm[0].position
        self.best_fitness = self.swarm[0].fitness

    def solve(self):
        cycle = 0

```

```

k = 0
prev_best = self.best_fitness
while cycle < 10 and k < 4000:
    quailies.append(self.best_fitness)

    self.best_sites = self.swarm[0:self.best_sites_count]
    self.sel_sites = self.swarm[self.best_sites_count:self.best_sites_count+self.sel_sites_count]

    bee_index = self.best_sites_count + self.sel_sites_count

    for i in range(len(self.best_sites)):
        for j in range(self.best_bee_count):
            self.swarm[bee_index].position = GenerateNeighborWay(
                self.best_sites[i].position)
            self.swarm[bee_index].fitness = fitness_func(self.swarm[bee_index].position)
            bee_index += 1

    for i in range(len(self.sel_sites)):
        for j in range(self.selected_bee_count):
            self.swarm[bee_index].position = GenerateNeighborWay(
                self.sel_sites[i].position)
            self.swarm[bee_index].fitness = fitness_func(self.swarm[bee_index].position)
            bee_index += 1

    for i in range(bee_index, self.beecount-bee_index):
        self.swarm[i] = Bee()

    self.swarm.sort(key=Bee.bee_sort)

    self.best_position = self.swarm[0].position
    self.best_fitness = self.swarm[0].fitness

    if prev_best > self.best_fitness:
        prev_best = self.best_fitness

    cycle = 0
    else:
        cycle += 1

    k += 1
class Bee:
    def __init__(self):
        """position = GenerateWay(X, depo)
        while position in population:
            position = GenerateWay(X, depo)
        population.append(position)"""
        position = GenerateWay(X, depo)
        fitness = fitness_func(position)

        self.position = position
        self.fitness = fitness

    def bee_sort(self):
        return self.fitness

start = time.monotonic()

```

```

hive = Hive(scout_bee_count,
            selected_bee_count,
            best_bee_count,
            sel_sites_count,
            best_sites_count)

hive.solve()

print(hive.best_fitness, hive.best_position)

import pandas as pd
import datetime
from backend.app.salesmanpar import best_way
from analytics.count_fulling_entities import count_fulling_entities
from db.dbi import db

def return_entities_for_shipment_by_date(date):
    df = list(db.get_entities_for_shipment_by_date(date)['ce_code'])
    return df

def return_plan(date):
    depo = 1
    df_entities = db.get_entities(depo=1)
    df_costs = db.get_costs()
    needed_entities = return_entities_for_shipment_by_date(date)

    X = list(
        df_entities
        [lambda x: (x['ce_code'].isin(needed_entities))]['ce_code']
    )

    X.append(depo)

    M = {}
    for i in X:
        temp = {}
        for j in X:
            if i == j:
                temp[j] = float('inf')

            else:
                temp[j] = int(df_costs[
                    (df_costs['from_code'] == i)
                    &
                    (df_costs['to_code'] == j)
                ]['distance'])

        M[i] = temp

    S, ib, result_way = best_way(M, X, depo)

    return S, ib, result_way

```

```

def get_plan_by_date(date):
    result_way = (
        db.get_plan_by_date(date).loc[0, 'point']
        if len(db.get_plan_by_date(date))
        else []
    )

    df_coords = db.get_coords()
    plan = []
    for i in result_way:
        plan.append({
            'lat': float(df_coords[df_coords['ce_code'] == i]['x_coord']),
            'lng': float(df_coords[df_coords['ce_code'] == i]['y_coord'])
        })

    return plan

def run_analytics_for_params(date):

    df = db.get_balances_by_date(date)
    entities = count_fulling_entities()
    print(df, entities)

    for i in range(2):
        for index, row in entities.iterrows():
            db.fill_balance(
                code=int(row['ce_code']),
                date=(pd.to_datetime(date)+datetime.timedelta(days=i+1)),
                percent=(row['percent']+row['intensity']*(i+1))
            )

    for index, row in entities.iterrows():
        reco_today = row['percent'] + row['intensity']
        reco_tomorrow = row['percent'] + row['intensity'] * 2

        if reco_today > 1:
            db.add_forecast_shipment(
                ce_code=row['ce_code'],
                next_shipment=(pd.to_datetime(date)+datetime.timedelta(days=1))
            )

        elif (reco_today < 1) and (reco_tomorrow > 1):
            if abs(reco_today - 1) > (reco_tomorrow - 1):
                db.add_forecast_shipment(
                    ce_code=row['ce_code'],
                    next_shipment=(pd.to_datetime(date) + datetime.timedelta(days=2))
                )
            else:
                db.add_forecast_shipment(
                    ce_code=row['ce_code'],
                    next_shipment=(pd.to_datetime(date) + datetime.timedelta(days=1))
                )

    for date in [pd.to_datetime(date) + datetime.timedelta(days=1),

```

					ДП 6117.00.000 ПЗ	Арк.
						80
Змн.	Арк.	№ докум.	Підпис	Дата		

```

pd.to_datetime(date) + datetime.timedelta(days=2)):
S, ib, result_way = return_plan(date)

if S != float('inf'):
    db.add_plan(date, result_way, S, S)

import pandas as pd
import googlemaps

API_key = 'AIzaSyAjJiOQYpv9x7CWjzfcHFkRPmMJBTy_3C0'
gmaps = googlemaps.Client(key=API_key)

df = pd.read_csv('./data/cash_entities/cash_entities.csv')

distance_df = pd.DataFrame({
    'FROM_CODE': [],
    'TO_CODE': [],
    'DISTANCE': [],
    'DURATION': []
})

origin_codes = []
dest_codes = []
distances = []
durations = []

for origin_index, origin_row in df.iterrows():
    origin_code = origin_row['CODE']
    origin_lat = origin_row['X_COORD']
    origin_lng = origin_row['Y_COORD']

    for dest_index, dest_row in df.iterrows():
        dest_code = dest_row['CODE']
        dest_lat = dest_row['X_COORD']
        dest_lng = dest_row['Y_COORD']

        result = gmaps.distance_matrix((origin_lat, origin_lng),
                                       (dest_lat, dest_lng),
                                       mode='driving')

        distance = result['rows'][0]['elements'][0]['distance']['value']
        duration = result['rows'][0]['elements'][0]['duration']['value']

        origin_codes.append(origin_code)
        dest_codes.append(dest_code)
        distances.append(distance)
        durations.append(duration)

distance_df['FROM_CODE'] = origin_codes
distance_df['TO_CODE'] = dest_codes
distance_df['DISTANCE'] = distances
distance_df['DURATION'] = durations
distance_df.to_csv('distances/distances.csv', sep=',', index=None)

```

```

import datetime
import csv
import random

from db.dbi import db

start_date = '2020-01-01'
end_date = '2020-05-25'

entities_df = db.get_atms()['ce_code']
entities = [i for i in entities_df]

entities_percents = [0.0 for _ in entities_df]
entities_intensities = [float(f'{random.random():.3f}') for _ in entities_df]

print(entities_intensities)

start = datetime.datetime.strptime(start_date, "%Y-%m-%d").date()
end = datetime.datetime.strptime(end_date, "%Y-%m-%d").date()

date_range = [
    start + datetime.timedelta(days=x)
    for x in range(0, (end-start).days)
]

for date in date_range:
    with open(f'balances/{date}.csv', 'w') as file:
        writer = csv.writer(file, delimiter=',')
        writer.writerow(['DATE', 'CODE', 'PERCENT'])
        for i in range(len(entities_df)):
            min_range = entities_intensities[i]-random.random()/10
            current_intensity = random.uniform(
                min_range if min_range > 0 else 0,
                entities_intensities[i]+random.random()/10
            )
            temp = entities_percents[i]+current_intensity
            intensity = float(f'{temp:.3f}') if temp < 1 else float(f'{temp-1:.3f}')
            entities_percents[i] = intensity
            writer.writerow([date, entities[i], intensity])

import datetime

from db.dbi import db

def count_fulling_entities():
    min_max_date = db.get_min_max_date()

    start = min_max_date.loc[0, 'min']
    end = min_max_date.loc[0, 'max']

    date_range = [
        start + datetime.timedelta(days=x)
        for x in range(0, (end-start).days+1)
    ]

```

Змн.	Арк.	№ докум.	Підпис	Дата

```

entities_df = db.get_atms()
entities_df['percent'] = 0
entities_df['current_days'] = 0
entities_df['intensity'] = 0

balances = db.get_balances()

for index, row in entities_df.iterrows():
    fulling_days = []
    for date in date_range:
        df = balances[balances['date'] == date]
        percent_by_date = float(df[df['code'] == row['ce_code']]['percent'])

        if entities_df.loc[index, 'percent'] <= percent_by_date:
            entities_df.loc[index, 'current_days'] += 1
        elif entities_df.loc[index, 'percent'] > percent_by_date:
            fulling_days.append(entities_df.loc[index, 'current_days'])
            entities_df.loc[index, 'current_days'] = 1

    entities_df.loc[index, 'percent'] = percent_by_date

    entities_df.loc[index, 'intensity'] = len(fulling_days)/sum(fulling_days)

return entities_df

```

```

import React, { Component, Fragment, useState } from 'react'
import { withStyles } from '@material-ui/core/styles';
import FormControlLabel from '@material-ui/core/FormControlLabel';
import Checkbox from '@material-ui/core/Checkbox';
import axios from 'axios'
import moment from "moment"

import DatePicker from 'react-datepicker'
import Dropzone from 'react-dropzone';

import 'react-datepicker/dist/react-datepicker.css'

import ShipmentsTable from './table'
import Map from './map'

import './styles.css'

import Graph1 from './graph1'
import Graph2 from './graph2'

const GreenCheckbox = withStyles({
  root: {
    color: "#0e6655",
    '& $checked': {
      color: "#0e6655",
    },
  },
  checked: {},
})

```



```
})(props) => <Checkbox color="default" {...props} />);
```

```
class Application extends Component {
```

```
  constructor(props) {
    super(props)
    this.state = {
      date: new Date(),
      center: {
        lat: 50.4250244,
        lng: 30.4503335
      },
      zoom: 11,
      cash_entities: [],
      shipments: [],
      plan: [],
      checkbox: false,
      files: []
    }
  }
}
```

```
  fetchAtms = async () => {
    const res = await axios.get('http://127.0.0.1:5002/intensity',
      {
        method: 'GET',
        mode: 'no-cors',

        params: {
          date: this.state.date
        },
        headers: {
          "Access-Control-Allow-Origin": "*",
          'Content-type': 'application/json'
        }
      }
    )
    this.setState({ cash_entities: res.data.cash_entities })
  }
```

```
  fetchShipments = async () => {
    const res = await axios.get('http://127.0.0.1:5002/shipments',
      {
        method: 'GET',
        mode: 'no-cors',
        headers: {
          "Access-Control-Allow-Origin": "*",
          'Content-type': 'application/json'
        }
      }
    )
    this.setState({ shipments: res.data.shipments })
  }
```

```
  fetchPlan = async () => {
    const res = await axios.get('http://127.0.0.1:5002/plan_by_date',
```

					ДП 6117.00.000 ПЗ	Арк.
						84
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    {
      method: 'GET',
      mode: 'no-cors',
      params: {
        date: this.state.date
      },
      headers: {
        "Access-Control-Allow-Origin": "*",
        'Content-type': 'application/json'
      }
    }
  )
  this.setState({ plan: res.data.plan })
  console.log(this.state)
}

runAnalytics = async () => {
  const res = await axios.get('http://127.0.0.1:5002/run_analytics',
    {
      method: 'GET',
      mode: 'no-cors',
      params: {
        date: this.state.date
      },
      headers: {
        "Access-Control-Allow-Origin": "*",
        'Content-type': 'application/json'
      }
    }
  )
  this.setState({ plan: res.data.plan })
  console.log(this.state)
}

componentDidMount() {
  this.fetchAtms()
  this.fetchShipments()
  this.fetchPlan()
}

update() {
  console.log(this.state.date)
  this.fetchAtms()
  this.fetchPlan()
}

handleDatePickerClick(date) {
  this.setState({ date: new Date(date) }, ()=>this.update())
}

handleDrop = acceptedFiles => this.setState({ files: acceptedFiles })

render() {
  return (
    <Fragment>

```

```

<div
  style={{
    background: "#0e6655",
    color: "#f7f9f9",
    height: "9vh",
    fontFamily: "Arial Black",
    fontSize: "5vh",
    paddingTop: "1vh",
    paddingLeft: "2vh"
  }}
>
  Bank Analytics
  <button
    style={{
      width: "10vw",
      height: "4vh",
      color: "#0e6655",
      background: "#f7f9f9",
      border: "none",
      borderRadius: "0.2rem",
      fontFamily: "Arial Black",
      fontSize: "1vw",

      marginLeft: "65vw"
    }}
    onClick={this.handleClick}>
    Upload Files
  </button>
</div>

<div>
  <div style={{ width: "40vw", height: "30vh", margin:"5vh", display: "inline-block"}}>
    <Graph1/>
  </div>
  <div style={{ width: "40vw", height: "30vh", margin:"5vh", display: "inline-block"}}>
    <Graph2/>
  </div>
</div>

{
  <div style={{ width: "50vw", height: "75vh", display: "inline-block", marginTop: "2.5vh",
marginRight: "2.5vh", marginLeft: "2.5vh"}}>
    <Map cash_entities={this.state.cash_entities} plan={this.state.plan}/>
  </div>

  <div style={{ display: "inline-block"}}>
    <div style={{ display: "inline-block"}}>
      <DatePicker
        inline
        selected={this.state.date}
        onChange={ (date) => this.handleDatePickerClick(date)}
        maxDate={ new Date(new Date().getFullYear(),new Date().getMonth(),new
Date().getDate()+5)}
      />
    </div>
  </div>
}

```

```

<div style={{ display: "inline-block" }}>
  <div style={{ paddingLeft: "1vw" }}>
    <FormControlLabel
      control={
        <GreenCheckbox
          checked={this.state.checkbox}
          onChange={() => this.setState({ checkbox: !this.state.checkbox })}
          name="checkedG"
        />
      }
      label="Every day"
    />
  </div>
  {
    moment(this.state.date).format('YYYY-MM-DD') === moment(new
Date()).format('YYYY-MM-DD')
    && <button
      style={{
        width: "10vw",
        height: "4vh",
        background: "#0e6655",

        color: "#f7f9f9",
        border: "none",
        borderRadius: "0.2rem",
        fontFamily: "Arial Black",
        fontSize: "1vw",
        margin: "1vw"

      }}
      onClick={this.runAnalytics}>
      Run Analytics
    </button>

  }
  <button
    style={{
      width: "10vw",
      height: "4vh",
      background: "#0e6655",
      color: "#f7f9f9",
      border: "none",
      borderRadius: "0.2rem",
      fontFamily: "Arial Black",
      fontSize: "1vw",
      margin: "1vw"

    }}
    onClick={this.handleClick}>
    Report
  </button>
</div>
<div style={{ width: "40vw", height: "50vh", margin: "2.5vh" }}>
  <ShipmentsTable rows={this.state.shipments} />
</div>
</div>

{<div>
  <Dropzone onDrop={this.handleDrop}>

```

```
{ ({ getRootProps, getInputProps }) => (  
  <div {...getRootProps({ className: "dropzone" })}>  
    <input {...getInputProps()} />  
    <p>Drag'n'drop files, or click to select files</p>  
  </div>  
)}  
</Dropzone>  
<div>  
  <strong>Files:</strong>  
  
  </div>  
</div>  
</Fragment>  
)  
}  
}
```

export default Application

					ДП 6117.00.000 ПЗ	Арк.
						88
Змн.	Арк.	№ докум.	Підпис	Дата		

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації і управління

УЗГОДЖЕНО

Керівник проєкту

_____ Олена ЖДАНОВА

(підпис)

(вл. ім'я, прізвище)

“13” квітня 2020 р.

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

(підпис)

(вл. ім'я, прізвище)

“14” квітня 2020 р.

Інформаційна система складання планів обслуговування терміналів
з прийому платежів у населення

ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП 6117.01.000 ТЗ*

на 8 сторінках

Київ – 2020 року

ЗМІСТ

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ.....	3
1.1 Повне найменування системи.....	3
1.2 Найменування організації-замовника та організації-учасників робіт.....	3
1.3 Перелік документів, на підставі яких створюється система.....	3
1.4 Планові терміни початку і закінчення роботи зі створення системи.....	3
2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ СИСТЕМИ.....	4
2.1 Призначення системи.....	4
2.2 Цілі створення системи.....	4
3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ.....	5
4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	6
4.1 Вимоги до функціональних характеристик.....	6
4.2 Вимоги до надійності.....	6
4.3 Вимоги до складу і параметрів технічних засобів.....	6
5 СТАДІЇ І ЕТАПИ РОЗРОБКИ.....	7
6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ.....	8

ДП 6117.01.000 ТЗ

					<i>ДП 6117.01.000 ТЗ</i>			
<i>Зм.</i>	<i>Арк.</i>	<i>Прізвище</i>	<i>Підпис</i>	<i>Дата</i>	Інформаційна система складання планів обслуговування терміналів з прийому платежів у населення	<i>Лім.</i>	<i>Лист</i>	<i>Листів</i>
<i>Розроб.</i>		<i>Лукова О.Ю.</i>					2	
<i>Перевірив.</i>		<i>Жданова О.Г.</i>						
<i>Н. кон.</i>		<i>Телишева Т.О.</i>				КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-61		
<i>Затв.</i>		<i>Павлов О.А.</i>						

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Повне найменування системи

Інформаційна система складання планів обслуговування терміналів з прийому платежів у населення.

1.2 Найменування організації-замовника та організацій-учасників робіт

Замовник системи: ТОВ «Українські інформаційні технології».

Розробник системи: студентка 4 курсу ФІОТ, кафедри АСОІУ Лукова Оксана Юріївна.

1.3 Перелік документів на підставі яких створюється система

Підставою для розробки системи є завдання на переддипломну практику.

1.4 Планові терміни початку і закінчення роботи зі створення системи

Плановий термін початку робіт над системою – 18.05.2020.

Плановий термін завершення робіт над системою – 01.06.2020.

					ДП 6117.01.000 ТЗ	Арк.
						92
Змн.	Арк.	№ докум.	Підпис	Дата		67

2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ СИСТЕМИ

2.1 Призначення системи

Складання планів інкасації терміналів з прийому платежів.

2.2 Цілі створення системи

Метою створення системи є зменшення витрат на інкасацію терміналів з прийому платежів за рахунок складання оптимальних або близьких до них планів інкасації терміналів та прогнозування наповненості терміналів.

					ДП 6117.01.000 ТЗ	93	Арк.
							4
Змн.	Арк.	№ докум.	Підпис	Дата			

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Об'єктом автоматизації є процес побудови маршрутів для інкасаторських машин та формування звітності про зменшення витрат завдяки використанню системи. Результат роботи досягається за декілька етапів:

- прогнозування наповненості терміналів готівкою, яка надходить від клієнтів – користувачів терміналів;
- складання планів інкасації терміналів з мінімальною вартістю реалізації, яке враховує результати прогнозування наповненості терміналів.

Оптимальність планів інкасації оцінюється вартістю проїзду за встановленими маршрутами руху інкасаційних машин.

Даних про автоматизацію цих процесів в Україні немає.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Система повинна виконувати наступні функції:

- ведення наявних терміналів;
- ведення обліку інкасаторських машин;
- прогнозування наповненості терміналів готівкою;
- складання планів інкасації терміналів;
- формування звітності.

4.2 Вимоги до надійності

Система повинна оброблювати всі виключні ситуації, що пов'язані з некоректними отриманими даними.

Усі дані у системі є конфіденційною інформацією і не можуть бути розголошеними.

4.3 Вимоги до складу і параметрів технічних засобів

Мінімальні вимоги до ЕОМ для коректної роботи системи:

- оперативна пам'ять – 1 Гб;
- жорсткий диск – 1 Гб;
- процесор – Intel Core i3 або краще.

Програмні засоби, що використовуються для проведення тестування:

- операційна система MacOS Catalina.

Технічні засоби, що використовуються для проведення тестування:

- оперативна пам'ять – 16 Гб;
- жорсткий диск – 500 Гб;
- процесор – Intel Core i7 2,7 ГГц;
- графічна карта - Intel Iris Plus Graphics 655.

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Таблиця 5.1 – Етапи розробки

№	Назва	Результат виконання
1.	Постановка задачі	Технічне завдання
2.	Розробка математичної моделі	Розроблена математична модель
3.	Генерація тестових даних	Розроблені файли вхідних даних
4.	Розробка алгоритму	Розроблений алгоритм
5.	Тестування алгоритму	Протестований алгоритм
6.	Розробка інтерфейсу	Розроблений інтерфейс
7.	Тестування інтерфейсу та алгоритму	Коректна робота інтерфейсу та алгоритму разом
8.	Оформлення ПЗ	Готова пояснювальна записка

Змн.	Арк.	№ докум.	Підпис	Дата

6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Перелік документів, що повинні бути представлені на контроль:

- технічне завдання;
- пояснювальна записка опису інформаційної системи складання планів обслуговування терміналів з прийому платежів у населення;
- інформаційна система складання планів обслуговування терміналів з прийому платежів у населення;
- тестові сценарії;
- керівництво користувача.

					ДП 6117.01.000 ТЗ	97	Арк.
							8
Змн.	Арк.	№ докум.	Підпис	Дата			

Власник документу:
Попенко Володимир Дмитрович

Дата перевірки:
04.06.2020 18:34:17 EEST

Дата звіту:
04.06.2020 20:16:20 EEST

ID перевірки:
1003783781

Тип перевірки:
Doc vs Internet + Library

ID користувача:
77149

Назва документу: Lukova_bachelor

ID файлу: 1003798235 Кількість сторінок: 55

Кількість слів: 7196 Кількість символів: 51267 Розмір файлу: 6.40 MB

7.09% Схожість

Найбільша схожість: 5.35% з джерело бібліотеки. ID файлу: 5882670

3.71% Схожість з Інтернет джерелами

2

Page 57

6.96% Текстові збіги по Бібліотеці акаунту

12

Page 57

0% Цитат

Не знайдено жодних цитат

0% Вилучень

Вилучений текст відсутній

Підміна символів

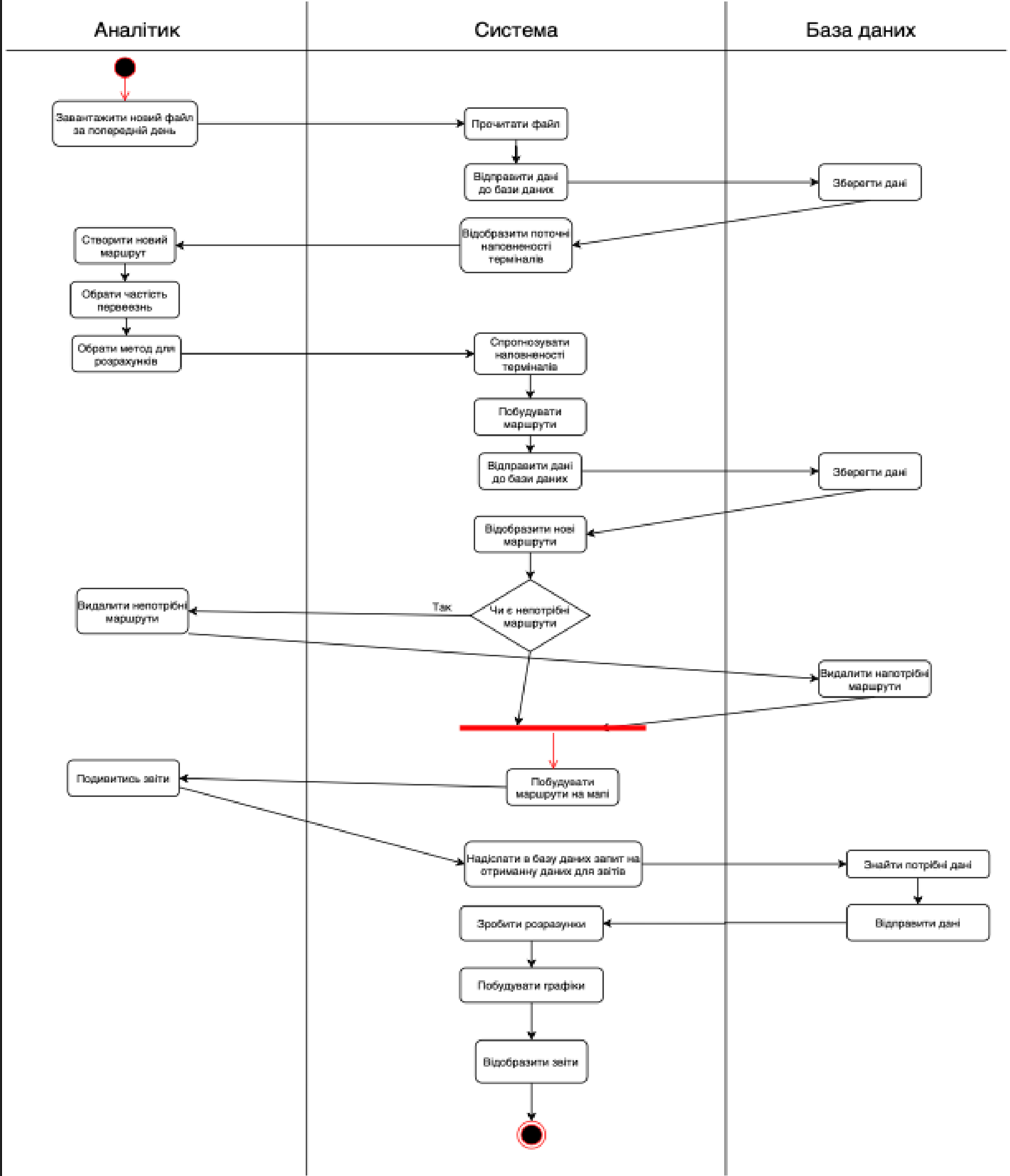
Заміна символів

13

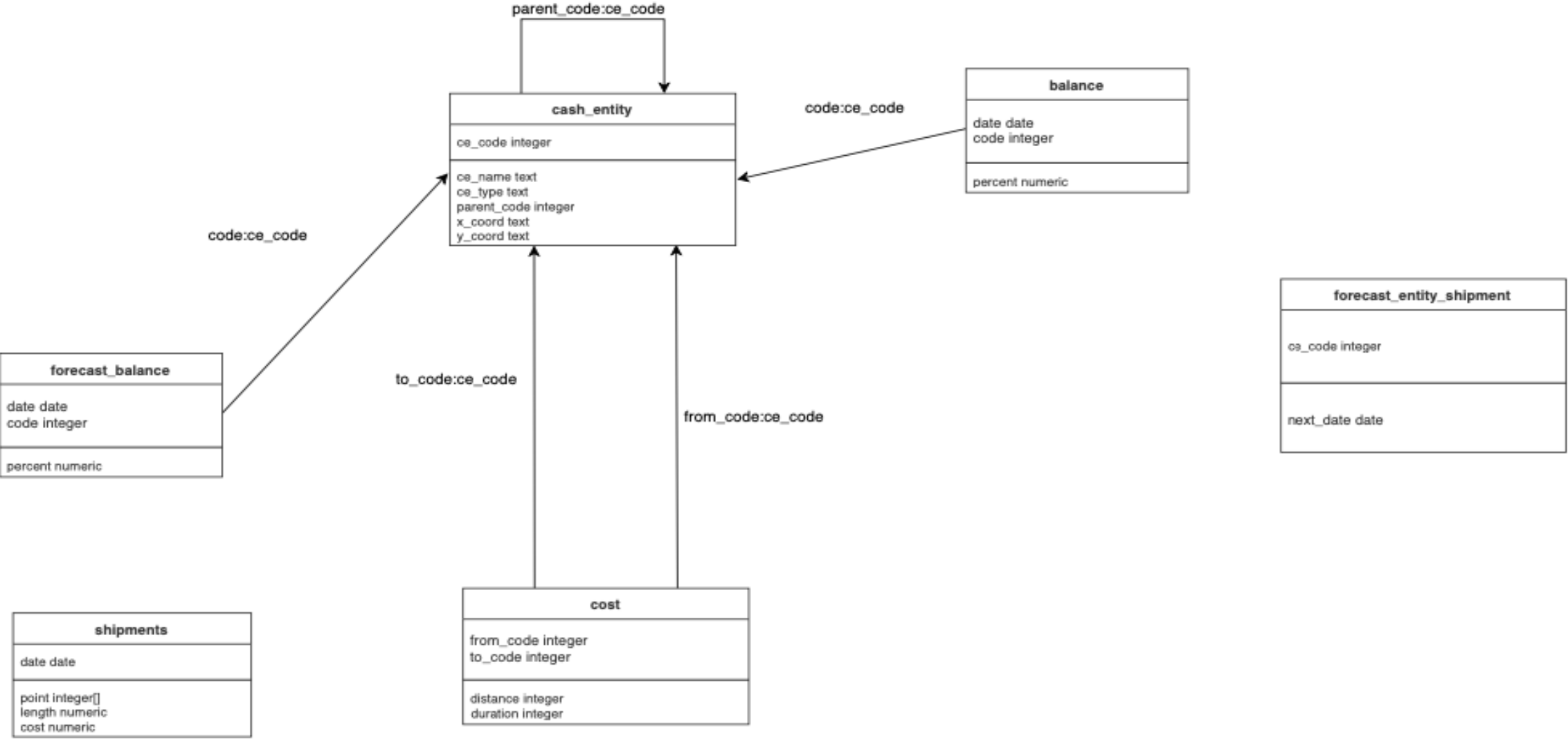
Графічний матеріал до дипломного проєкту

на тему: Інформаційна система складання планів обслуговування
терміналів з прийому платежів у населення

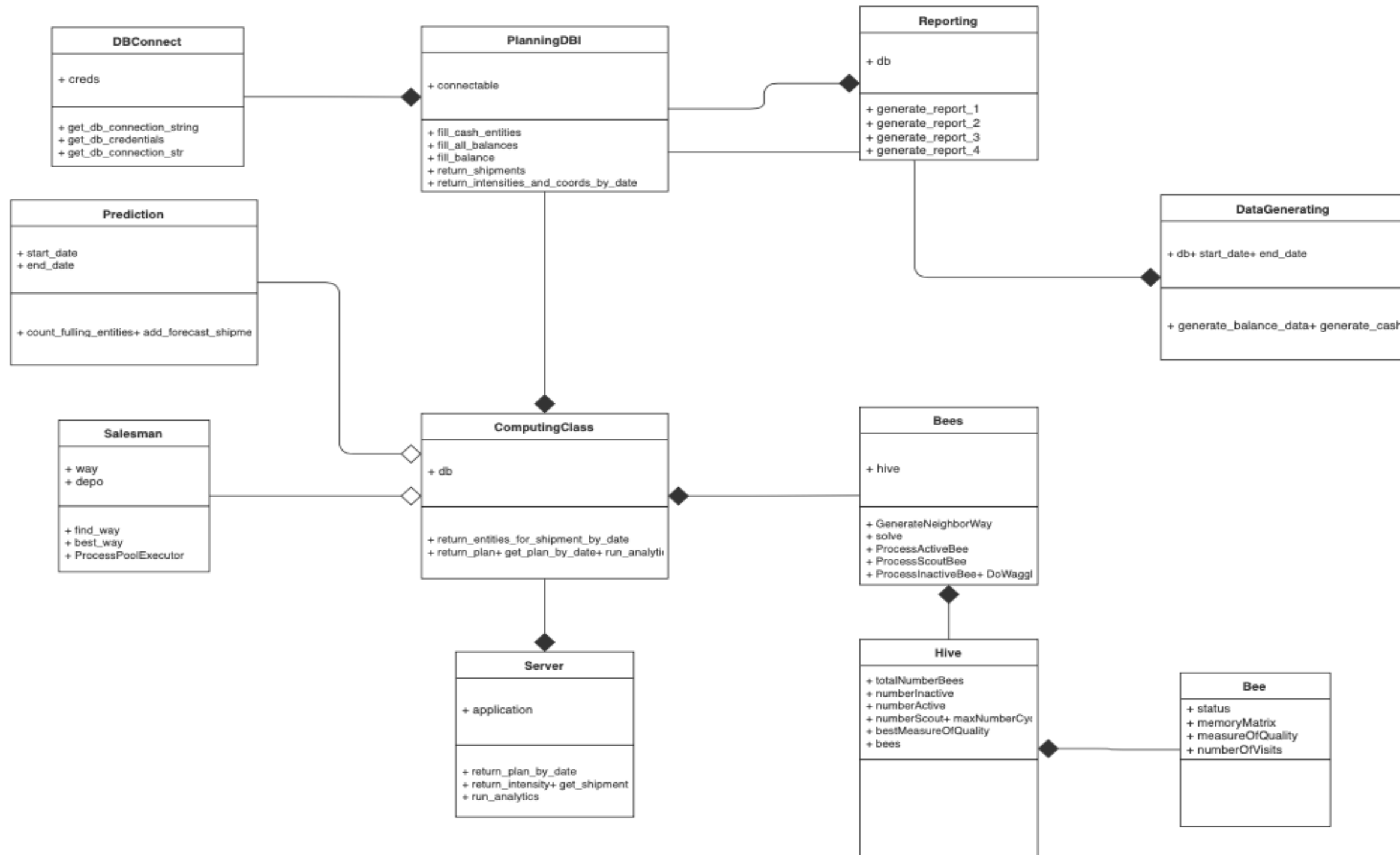
Київ – 2020 року



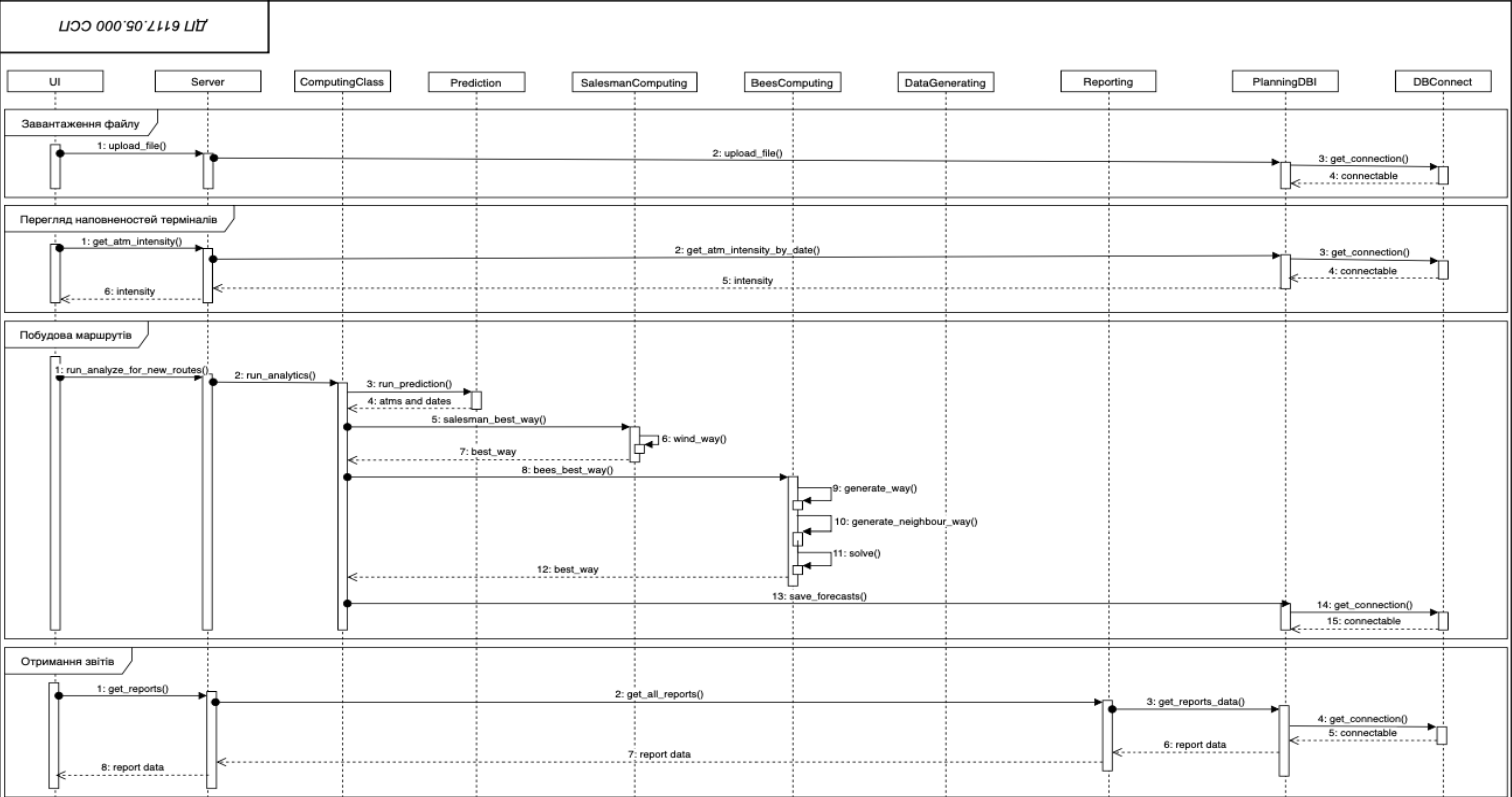
						ДП 6117.02.000 ССД			
						Схема структурна діяльності	Доп.	Міся	Місяць
Зм.	Апр.	№ докум.	Підп.	Дата					
Розроб.		Пуксєв О.Ю.							
Прав.		Жданова О.Г.							
Т. Ком.							Аркуш 1		Аркушів 1
Н. Ком.		Телищєва Т.О.				Інформаційна система складання звітів обслуговування терміналів з прийому платежів у касових	КПІ ім. Івора Сікорського кафедра АСОІУ ар. ІС-81		
Зам.		Жданова О.Г.							



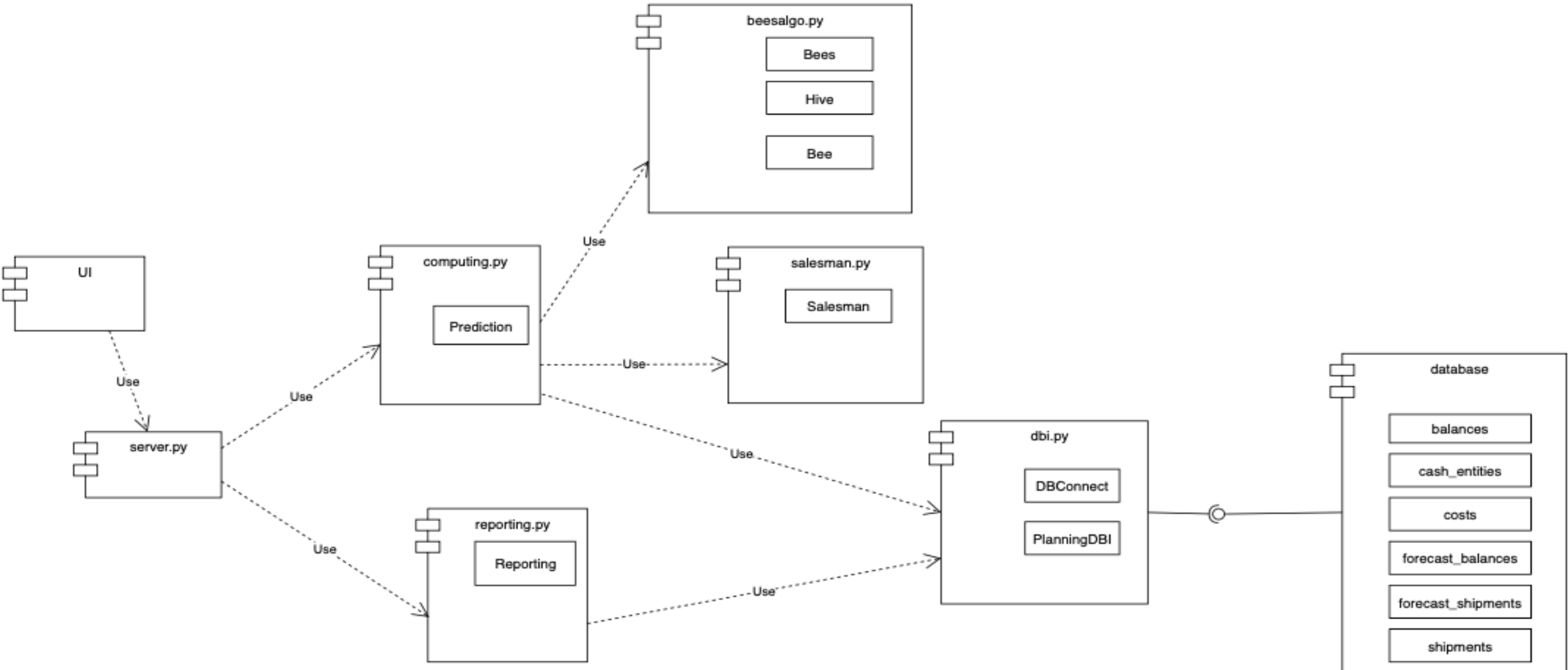
					ДП 6117.03.000 СБД						
					Схема бази даних	Літера		Маса		Масштаб	
Зм.	Арк.	№ документа	Підпис	Дата							
Розробив	Лукова О.Ю.										
Перевірів	Жданова О.Г.										
Т. кон.											
						Аркуш 1		Аркушів 1			
Н. кон.	Телишева Т.О.				Інформаційна система складання планів обслуговування	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-61					
Затвердив	Жданова О.Г.				терміналів з прийому платежів у населення						



					ДП 6117.04.000 ССК				
					Схема структурна класів програмного забезпечення	Літера		Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата					
Розробив		Лукова О.Ю.							
Перевірів		Жданова О.Г.							
Т. кон.									
					Інформаційна система складання планів обслуговування терміналів з прийому платежів у населення	Аркуш 1		Аркушів 1	
Н. кон.		Телишева Т.О.				КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-61			
Затвердив		Жданова О.Г.							



						ДП 6117.05.000 ССП				
						Схема структурна послідовності	Літера		Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата						
Розробив		Лукова О.Ю.								
Перевірів		Жданова О.Г.								
Т. кон.										
							Аркуш 1		Аркушів 1	
Н. кон.		Телишева Т.О.				Інформаційна система складання планів обслуговування терміналів з прийому платежів у населення	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-61			
Затвердив		Жданова О.Г.								



					ДП 6117.06.000 ССК			
					Схема структурна компонентів програмного забезпечення	Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата				
Розробив		Лукова О.Ю.						
Перевірів		Жданова О.Г.						
Т. кон.					Інформаційна система складання планів обслуговування терміналів з прийому платежів у населення	Аркуш 1		Аркушів 1
Н. кон.		Телишева Т.О.				КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-61		
Затвердив		Жданова О.Г.						